NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

WEB DATABASE DEVELOPMENT

by

Nikolaos A. Tsardas

September 2001

Thesis Advisor: Thomas Wu
Second Reader: Chris Eagle

Approved for public release; distribution is unlimited

Report Documentation Page

Report Date 30 Sep 2001	**Report Type** N/A	**Dates Covered (from... to)** -

Title and Subtitle Web Database Development	**Contract Number**
	Grant Number
	Program Element Number
Author(s) Nikolaos Tsardas	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Research Office Naval Postgraduate School Monterey Ca. 93943-5138	**Performing Organization Report Number**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Sponsor/Monitor’s Acronym(s)**
	Sponsor/Monitor’s Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	
Abstract	
Subject Terms	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 91	

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Web Database Development	**5. FUNDING NUMBERS**
6. AUTHOR(S) : Nikolaos Tsardas	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

ABSTRACT (maximum 200 words)

This thesis explores the concept of Web Database Development using Active Server Pages (ASP) and Java Server Pages (JSP). These are among the leading technologies in the web database development. The focus of this thesis was to analyze and compare the ASP and JSP technologies, exposing their capabilities, limitations, and differences between them. Specifically, issues related to back-end connectivity using Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC), application architecture, performance, and web security were examined.

For demonstration purposes, two applications were developed, one with ASP and another with JSP. The user interface and the functionality of these two applications were identical, while the architecture, performance, and back-end connectivity was totally different.

13. SUBJECT TERMS Web Database Development, Active Server Pages, Java Server Pages, ODBC, JDBC, Web Servers, Servlets, JavaBeans, Multi-Tier Architecture, IIS, Tomcat, Java, VBScript	**15. NUMBER OF PAGES** 98
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

WEB DATABSE DEVELOPMENT

Nikolaos A. Tsardas
Captain, Hellenic Army
B.S., Hellenic Army Academy, 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2001

Author: ______________________________
Nikolaos A. Tsardas

Approved by: ______________________________
Thomas Wu, Thesis Advisor

____
Chris Eagle, Second Reader

____
Chris Eagle, Chairman
Department of Computer Science and Information System & Technology

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis explores the concept of Web Database Development using Active Server Pages (ASP) and Java Server Pages (JSP). These are among the leading technologies in the web database development. The focus of this thesis was to analyze and compare the ASP and JSP technologies, exposing their capabilities, limitations, and differences between them. Specifically, issues related to back-end connectivity using Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC), application architecture, performance, and web security were examined.

For demonstration purposes, two applications were developed, one with ASP and another with JSP. The user interface and the functionality of these two applications were identical, while the architecture, performance, and back-end connectivity was totally different.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank my thesis advisor Prof. Thomas Wu and Prof. Chris Eagle for their support and guidance they gave me during this thesis.

Also I would like to thank my wife for her endless encouragement and patience.

I. INTRODUCTION

With the rapid development of web technology the traditional client-server model in database applications tends to lose its previous usefulness and popularity. A new model, based on web technology, introduces the multi-tier architecture in all applications that are used in Internet and Intranet networks. The new model of client is "thin", using mainly a "browser" to connect to the desired server. However, now between the client and the server is introduced a number of middle tiers who is role is to accept, filter, direct, and serve the data from the back-end database server to the client.

This thesis will explore this relatively new web technology and will evaluate its different implementations. This new technology will be used to replace existing database applications in the Hellenic army that use the traditional two-tier client-server architecture.

Presently, those existing database applications are not very flexible. They involve a central database system and a number of "thick" clients. These clients are specific programs (usually from the same vendor of the database system) that are able to connect and retrieve or update information from the database server. A portion of the "application logic" is on the client side. This is a main reason that makes the management of these applications difficult. Whenever some changes need to be made to the logic, all clients are involved.

Performance is another issue. Usually these types of "thick" clients establish from the beginning a session with the server and keep that session open continuously, no matter if they actually use that session or not. When many clients are connected to the server the overall performance is reduced dramatically. On the other hand in a multi-tier application the web server is responsible for establishing each session when needed or terminating it when it is idle.

With the new model, all the logic is implemented either in the middle tiers (usually a web server) or in the database server. This makes administration much easier. Scalability is another issue, because web-technology permits users to increase to several thousands, or more, without any special administrative effort. Also the new technology permits any number of intermediate tiers that can implement specific tasks like security, indexing, and transaction control. Last, but not least this new architecture permits the connection of these database applications to the Internet with a minimum of effort (although security issues introduce some limitations).

Two of the leading technologies in web database development will be examined:

- Active Server Pages(ASP) from Microsoft
- Java Server Pages(JSP) from Java Platform

The objective of this thesis is to provide a comprehensive evaluation of these two different approaches in web database development. The conclusions will contribute to the actual transition from older models (client-server applications) to the most suitable and promising implementation of web technology (multi-tier applications). This transition will require a lot of resources and manpower, but in the long run it will be more flexible and more secure, it will save administrative resources and will exploit the Internet technology. The outline of my thesis is as follows:

In Chapter II, I present the back-end connectivity issues. ASP uses Open Database Connectivity (ODBC) while JSP uses Java Database Connectivity (JDBC). These two different connectivity techniques are presented and compared, because they are an important factor in the overall application's performance and functionality.

In Chapter III, I analyze and compare the built-in objects of these two technologies. These objects provide information related to the HTTP protocol, the client and the web server, facilitating the development effort. Every web application makes extensive use of these built-in objects.

In Chapter IV, I present the architecture issues. There is more than one available architectural model for each technology. These models are implemented differently by each technology. Architecture greatly affects maintenance efforts and overall application lifecycle.

In Chapter V, I analyze performance issues. I present specific factors that increase application performance for each technology. Performance results, from testing implementations, are presented and commented.

In Chapter VI, I present how each technology implements web security. Capabilities and limitations are described for each case. Web server issues are, also introduced.

Finally, in Chapter VII, I present the overall conclusions from the analysis and comparison of the ASP and JSP technologies.

THIS PAGE INTENTIONALLY LEFT BLANK

II. ODBC – JDBC Analysis and Comparison

In order to proceed with a detailed and thorough comparison between the ASP and JSP technologies we have to examine the underlying level that consists of the ODBC, JDBC or both.

A. ODBC

ODBC allows a single application to access different database management systems (DBMSs) using the same Application Programming Interface (API). It stands between the application and the Database Management System (DBMS). As you can see in Figure 1, the application's requests are passed to the ODBC driver manager. Then, the driver manager searches for the specific database driver. If that specific database driver isn't found then the process ends. If it is found, then the request is passed to that database driver in order to communicate with the database and the requesting process. The results (if any) are routed back from the database to the application through the specific database driver and the ODBC driver manager.

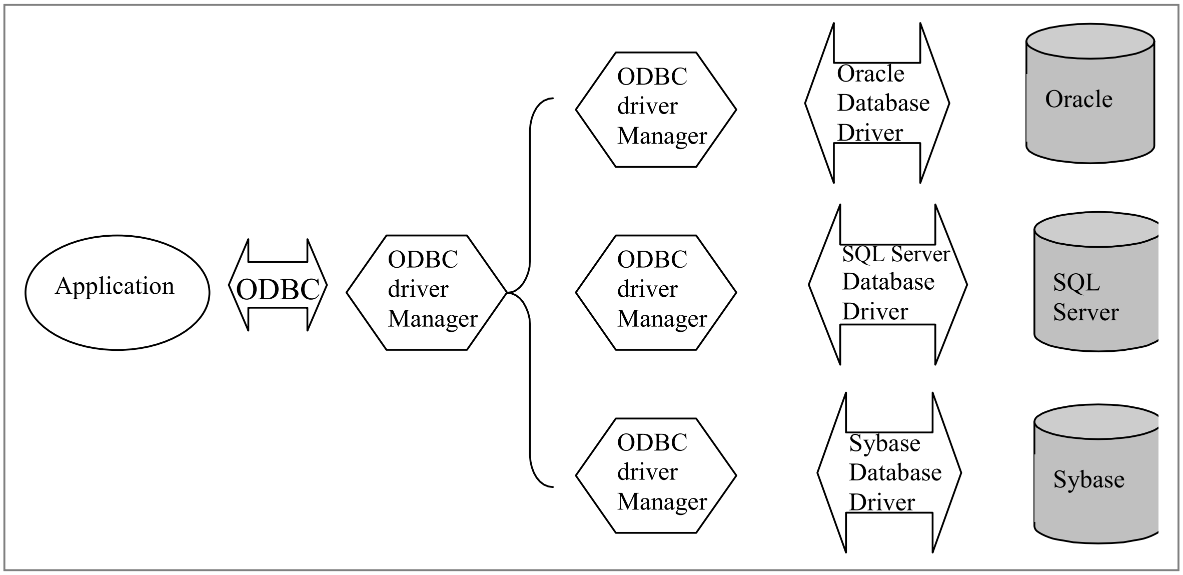


Figure 1. ODBC Model.

ODBC is not an easy to use programming interface. In order to simplify the process of using ODBC API Microsoft established the ActiveX Data Objects (ADO).

ADO is an API that includes all the functionality of the ODBC model, being both flexible and easy to program. ADO objects provide all the necessary interface to the programmer in order to access almost any data source (DBMS) for 1 – to n-tier client/server and web-based data-driven development.

ADO objects -in version 2.6- are relatively few in number as we can see in the following table [Ref. 1]:

Object	Description
Connection	The highest-level ADO object. Provides a pathway that other objects can use to access a database provider. It needs a data source name (DSN) or a connection string
Recordset	Provides access to the result set of a query or other database operation. It has numerous methods for the result set manipulation.
Field	Provides access to individual fields in the current record of a Recordset. ADO automatically creates Field objects when it creates Recordsets.
Property	Provides access to any characteristic of an ADO object. ADO automatically creates Property objects when it creates any type of object.
Command	Holds a command-and optionally command parameters-that will execute through a database connection. Mainly supports functions that the Record object doesn't: parameter queries and commands that don't produce a recordset.
Parameter	Holds a named value that the database provider will merge into a predefined query or stored procedure.
Error	Contains information about errors reported by a database provider.
Record	Represent one row of a recordset, or a directory or file in a file system.
Stream	Represents a binary stream of data.

Table 1. ODBC Model.

Although all the ADO objects are needed in a web database application, the most frequently used object is Recordset. This object is used for operations such as opening tables or queries, finding records, and displaying or updating field values. It has many methods and properties, but most web database work involves only a fraction of the Recordset object’s many capabilities. The most common method operations are:

- Opening and closing a recordset
- Moving the cursor forward or backward
- Finding a record that satisfies a specified condition
- Adding, deleting or updating a record
- Canceling any updates made to the current record or to a new record
- Making or canceling batch updates to a group of records
- Saving the recordset in a file or a Stream object
- Returns the recordset as a string
- Obtains database schema information from the provider

Also using the ADO’s Recordset and Field properties we can retrieve useful information -metadata- about the result set such as:

- The name and the value of each field in a record
- The type and the size of each field
- The maximum length of each field in bytes

Finally, depending on the supplied parameters in the Recordset’s Open method the result set can be opened in a number of ways:

- Statically: the recordset doesn’t reflects updates made from other users
- Dynamically: the recordset reflects updates made from other users
- Forward-Only: similar to static except that it can only scroll forward. Improves performance.
- Read-Only
- Locking records either from the time that the record is being edited or by the time that the update method is being called
- Allowing or not batch updates - transaction support

B. JDBC

JDBC is a collection of Java classes that provides the connectivity between the Java programs and the JDBC-enabled data sources. JDBC was derived from ODBC. The most common data source is a relational DBMS (RDBMS). Like ODBC, JDBC uses method or function calls to access its features, so it is also a call-level interface. With JDBC the programmer can have a standard interface to all DBMSs. Switching from one DBMS to another requires little or no changes in the application. Usually only the data type names and support for certain operation types are needed some changes. In that case, metadata can be used to solve that problem. The basic JDBC operations are:

- Load the JDBC specific driver for the DBMS (provided by the vendor)
- Open a connection using the DriverManager class. This step involves the previous loaded driver and the data source to be used
- Issue SQL statements to the DBMS through the connection
- Process result sets returned by the SQL statements

The JDBC API is contained in the java.sql and javax.sql packages. It consists mainly of interfaces rather than concrete classes because each vendor's implementation is specific to their particular DBMS protocol. Every vendor supplies a driver Java class that implements the java.sql.Driver interface. All these specific drivers along with the DriverManager class compose the middle layer of the JDBC, which primary function is to connect to a database and return a java.sql.Connection object. There are four types of JDBC drivers[Ref. 6]:

- Type 1 – JDBC-ODBC bridge. Drivers of this type connect to databases through an intermediate ODBC driver. Several drawbacks are involved with this approach, so Sun describes it as being experimental and appropriate for use only where no other driver is available. Both Microsoft and Sun provide that type of driver.
- Type 2 – Native API, partly Java. Similar to JDBC-ODBC bridge, type 2 drivers use native methods to call vendor specific API functions. These drivers are also subject to the same limitations as the JDBC-ODBC

bridge, in that they require native library files to be installed to client systems, which must be configured to use them.

- Type 3 – Pure Java to database middleware. Type 3 drivers communicate using a network protocol to a middleware server, which, in turn, communicates to one or more DBMSs.
- Type 4 – Pure Java direct to database. Drivers of this type call directly into the native protocol used by the DBMS.

Each of the four driver types has its own performance characteristics, but the API is exactly the same in all four cases.

The JDBC-ODBC bridge driver is limited to the capabilities of the underlying ODBC driver, which is single threaded and may perform poorly under a heavy load. Also it requires native code library *JdbcOdbc.dll* to be installed on the client system. Finally the JDBC-ODBC bridge driver requires an ODBC data source to be configured. Sun recommends the bridge should only be used for experimental purposes when no other JDBC driver is available.

On the other hand, the JDBC-ODBC bridge offers several significant advantages. ODBC is widely supported, so using the bridge makes possible accessing a wide variety of existing systems for which data sources are already configured. So, ODBC-enabled database products, such as Microsoft Access or FoxPro, are widely available. These features make the JDBC-ODBC bridge a good choice for low-volume web applications and a useful tool for learning JDBC. To use the JDBC-ODBC bridge in a Java application, a suitable ODBC data source must be configured. On Windows systems, this is done through the control panel ODBC Data Sources application. The data source should be configured as System DSN, because the JSP engine is typically running under a system user profile.

Type 2 driver consists of an interface, written partly in Java, between Java programs and the vendor specific database access middleware (for instance, Oracle SQL*Net). A type 2 JDBC driver is typically a direct bridge to the proprietary call-level API of the database product (Oracle OCI, for example). It doesn't require the presence of ODBC in the client. It does, however, require that administrators install and configure the database vendor’s proprietary database access middleware. Consequently, type 2 drivers cannot be used for the Internet.

Type 3 driver is a pure Java driver that, at the client, translates JDBC calls into a database-independent network protocol. At the server, a separate driver component translates the database –independent JDBC requests into database-specific, native calls. As a result, type 3 drivers are able to connect Java-based clients to whatever types of databases a separate server-side driver has been developed for. Type 3 drivers require basic network connectivity at the client(a TCP/IP protocol stack), but they do not rely on the presence of vendor-specific middleware or ODBC. However, type 3 drivers need to be sophisticated enough to handle a variety of networking situations, including firewalls. Also a Java program using a type 3 driver can claim to be highly generic because it will run on any Java-enabled platform with a TCP/IP connection to a database server. This type of driver is well-suited for use over the Internet. Typically, it provides support for features such as caching (connections, query results, etc), load balancing and advanced system administration such as logging and auditing. However, traversing the recordset may take longer, since the data comes through the back-end server.

The all-Java driver type 4 converts JDBC calls into the vendor specific DBMS protocol so that client applications can communicate directly with the database server. These types of drivers are completely implemented in Java to achieve platform independence and eliminate deployment administration issues. Since type 4 drivers don’t have to translate database requests to ODBC or a native connectivity interface or to pass the request on to another server, performance is typically quite good. Moreover, it boasts better performance than types 1 and 2. Also, there is no need to install special software on the client or server. Further, these drivers can be downloaded dynamically. However, using that type of driver, the user needs a different driver for each database.

The java.sql API contains 16 interfaces, 8 classes and 4 exception types. The javax.sql API adds another 12 interfaces and 2 classes. Many of these classes are of interest primarily to JDBC driver developers. The most commonly used classes are:

Class	Package	Description
Connection	java.sql	An active link to a database through which a Java program can read and write data, as well as explore the database structure and capabilities.
Statement	java.sql	An interface that allows SQL statements to be sent through a connection and retrieves the result sets. Used to execute static SQL statements
PreparedStatement	java.sql	An extension of Statement that uses precompiled SQL, possibly with dynamically set input parameters.
CallableStatements	java.sql	An extension of PreparedStatement that can invoke a stored procedure, if the DBMS support it.
ResultSet	java.sql	An ordered set of table rows produced by an SQL query or a call to certain metadata functions
DatabaseMetadata	java.sql	An interface containing numerous methods that provide information about the structure and capabilities of a database
ResultSetMetadata	java.sql	An interface that describes the columns of a ResultSet. It contains methods that describe the number of columns, each column's name, display size, data type, and class name.
DriverManager	java.sql	An interface that registers JDBC drivers and supplies connections that can handle specific JDBC URLs
SQLException	java.sql	The base exception class used by the JDBC API

Table 2. JDBC Classes.

JDBC version 2.0 introduced some new and important features, which include:

- **DataSource**. JDBC driver names and URLs can be stored in a name service and retrieved using Java Naming and Directory Interface (JNDI)
- **Connection Pooling**. A data source provider can offer connection pooling, allowing connections to be activated and recycled, usually with a significant performance improvement. This capability is configured entirely in the naming service and requires no changes to applications.
- **Scrollable result sets**. New methods are provided for forward and backward navigation, as well as relative and absolute cursor positioning

- **RowSets**. This new interface extends and generalizes java.sql.ResultSet so it can be detached from its database connection. This interface can be useful in JavaBeans model, with XML documents and for Personal Digital Assistant(PDA) applications.
- **BatchUpdates**. Transactions can be grouped and sent to the database as a unit, using the Connection's commit and rollback methods along with the Statement's addBatch and executeBatch methods.

Also, the first public draft of JDBC 3.0 was released for public review recently. Its new features include:

- Enhanced control of commit /rollback transaction boundaries.
- Configurability for connection pools
- Better interface to parameters in prepared and callable statements.

C. ODBC – JDBC COMPARISON

1. Simplicity

One of the stated goals for both ADO (using ODBC) and JDBC API was that they should be simple and easy to master. I think that both technologies achieved that goal. As we can see in the above tables, both of them use mainly nine objects. These objects have similar functionality. A programmer has to learn only these objects and three or four of their main methods in order to use them effectively.

2. Functionality

Initially, JDBC 1.0 was limited in its functionality. It had no support for either scrollable result sets or batch updates. JDBC 2.0 however, introduced a number of useful new features as shown above. Now JDBC and ADO have similar functionality as shown below:

Action	ADO (using ODBC)	JDBC	REMARKS
Connect to a database	Connection object using as parameters a data source name or connection string	Connection object using: -DriverManager class, with the database URL as a parameter or	Similar functionality for both.

Action			
		-A DataSource class from JNDI	
Get result sets from the database	Recordset object using as parameters a SQL string, an active connection object, a cursor type and a lock type	ResultSet object using the Statement object and an SQL string. The Statement object defines the active connection and the cursor type and a lock type	Similar functionality for both.
Result set's movement type between records	Allows any type of movement using Recordset's movement methods	Allows any type of movement using ResultSet's movement methods	Similar functionality for both.
Result set's updates without using SQL commands	Recordset's methods AddNew, Update, CancelUpdate, Delete	ResultSet's methods insertRow, updateRow, cancelRowUpdates, deleteRow	Similar functionality for both.
Result set's cursor types	Static, Dynamic, Forward-only and Keyset	Insensitive, Sensitive and Forward only	They both can open a recordset allowing all types of cursor movements or forward-only, reflecting updates or not. However ADO has richer variety.
Result set's lock type	Read-only, Pessimistic, Optimistic, BatchOptimistic	Read-only, Updatable	Both can open a record set either as read-only or allowing updates. However with ADO you can define also how an edited record can be locked during updates (from the beginning or when the update command is

Action			
			invoked)
Transactions	Using BatchOptimistic lock type permits the UpdateBatch and CancelBatch methods to be used for batch updates	Using Statement object methods addBatch, executeBatch and clearBatch	Similar functionality for both
Stored procedures or parameter queries	Command object with the appropriate parameters	CallableStatement and PreparedStatement object are used for stored procedures and parameter queries respectively	Similar functionality for both
Metadata	Recordset Properties and Field Properties provide adequate information about the database	The DatabaseMetaData and ResultSetMetaData objects have 170 methods that describe almost anything about the database	Similar functionality for both
Streams	Stream object provides streams of binary or text data	All Java's I/O classes are available. Also the Serializable interface allows the user to define streams with objects	JDBC has richer options using streams because it allows object serialization
Connection pooling	ODBC allows the connection's time out to be configured via the control panel of the Windows	JNDI allows full configuration of the connection pooling (pre-allocation of connections), based on the data source provider capabilities	JDBC uses connection pooling more effective- through Java objects- than ODBC does.

Table 3. ODBC-JDBC Comparison.

3. Performance

For JDBC drivers written in Java, performance can be an issue. In Type 1 the performance is degraded since the JDBC call goes through the bridge to the ODBC driver and then to the native database connectivity interface. The result comes back through the reversed process. In Type 2 drivers, ODBC is not used, so it performs better than Type 1 drivers. Types 3 and 4 are pure Java drivers and perform better than the other two driver types.

Concluding, ODBC (via ADO) and JDBC have generally similar features. Although there are pros and cons between them (multithreading, serialization, etc), the main functionality remains the same. An important factor in choosing one of the two technologies would be the platform on which they would be used. If we want to work in a cross-platform environment then JDBC is our only choice.

THIS PAGE INTENTIONALLY LEFT BLANK

III. ASP-JSP Objects Analysis

Server side programming like ASP or JSP needs a set of tools in order to receive and process information sent by the client or to send processed information back to the client. This set of tools is provided by built-in objects that are available in both technologies. ASP provides a set of objects named built-in objects, while JSP provides an equivalent set named implicit objects. Specifically, each of these sets of objects can be used to collect information about the user and his request, manage the transmission and presentation of that information, store the information for later use, respond to the user, and keep track of session-wide, application-wide, and web server-wide information. In fact, these objects are provided by the web server, the IIS web server in the ASP case, and the JSP container in the JSP case.

Each ASP or JSP developer has to use, more or less, that set of objects. Fortunately, we don't need to know much about what goes on inside of these objects. We can treat them as black boxes, using programming languages (VBScript or JavaScript for ASP and Java for JSP) to get information into them and out of them. We manipulate these objects using each object's API.

ASP built-in object API provides methods, properties, collections, and events:

- Methods are used to pass information to an object, have it perform some action, or return some value.
- Properties are the object's attributes (class or instance variables in the programming parlance), which describe that object and can be read or written, depending on the type of the property.
- Collections are groups of similar object properties, like the *ServerVariables* collection of the *Request* object that includes all the properties that describe the web server. Collections were created to facilitate development tasks (providing features like enumeration, and the *Count* property, among others).
- Events provide a way to define an action when an object's particular event occurs (like the start or termination of the *application* object).

JSP implicit object API provides methods and variables:

- Methods are used to pass information to an object, have it perform some action, or return some value.
- Variables are regular Java class or instance variables.

Knowing the capabilities and limitations of these objects is crucial for the web developer. We will evaluate these objects, by examining their functionality through their methods, properties, collections, events, and variables.

A. ASP BUILT-IN OBJECTS

The ASP engine, the *asp.dll* file, includes a foundation set of seven objects, providing access to information sent over HTTP, and can be extended through the addition of other objects. The core set of objects is often enough to build simple applications. Figure 2, shows graphically the relationship among the ASP built-in objects.

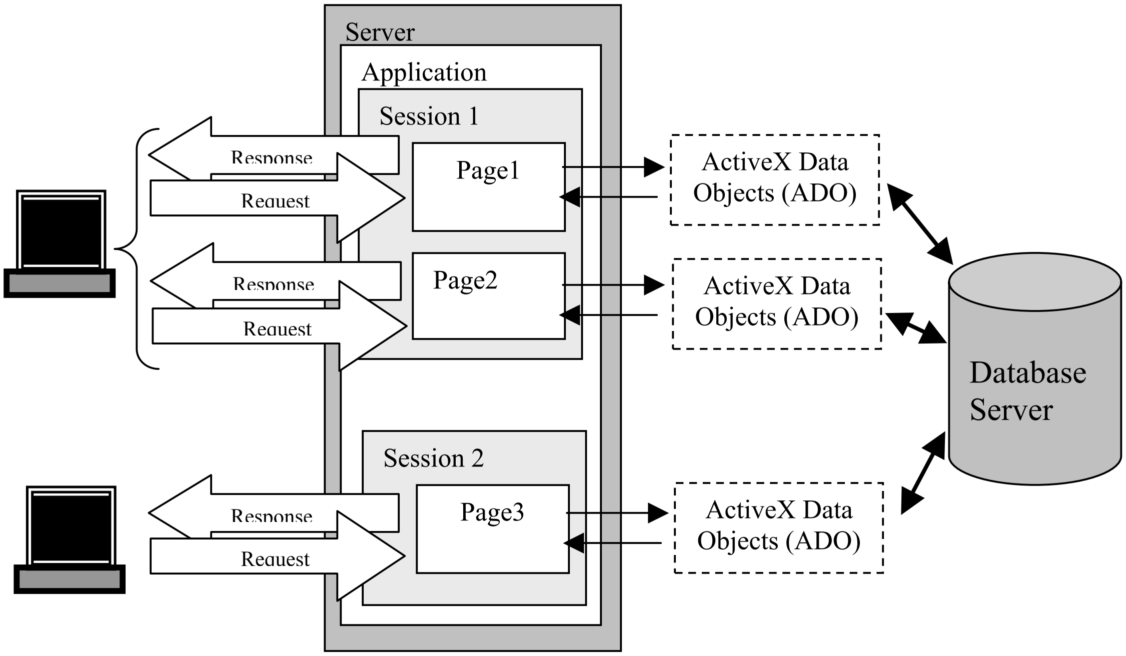


Figure 2. ASP Built-In Objects.

Microsoft has built a large number of components beyond the core ASP objects. Some like the ActiveX Data Objects are included with ASP, though they need separate initialization. A few, like the File Access components are installed as part of the scripting environment that comes with ASP, though they aren't ASP-specific, while others require a separate download and installation process.

A detailed analysis of the built-in ASP objects follows:

1. Request Object

The Request object contains information about the client's request that triggered the ASP page. This object supplies most of its information through collections, which represent the contents of HTTP headers, information from HTML forms, cookies, and other material. We can easily use collections with *VBScript* in the following way:

variable = object.collection(property_name)

e.g. cookieID = Request.Cookies("id")

The following collections, properties and methods are provided:

Category	Name	Description
Collections	Client Certificate	If the browser submits a digital certificate along with the normal HTTP request, then the collection will contain an entry for each field in the certificate (issuer, user, serial number, validity period, etc).
Collections	Forms	Contains any values the browser transmitted from an HTML form using the POST method. Most HTML forms contain one or more Submit buttons plus a number of text boxes, selection lists, etc. Each of these form elements has a name and a value. When the client clicks the Submit button, the browser sends all the form element names and values to a URL specified in the form's action attribute. If that action attribute specifies an ASP page using the POST method then all the form's element values appear in the Request.Form collection.
Collections	Cookies	Contain one entry for each cookie value the Web server sent to the client's browser as part of an earlier response. This is one solution to the problem of Web transactions being stateless.

Category		
Collections	QueryString	It is used when the client submits an HTML form with the GET method or a URL containing at the end a query string. However, it increases the length of the URL. If the URL exceeds the capacity of either the browser or the Web server, this can lead to loss of data with unpredictable results.
Collections	Server-Variables	Contain information about the visitor, the browser, the Web server and other details depending on the circumstances. One useful value is the "HTTP_USER_AGENT" string, which identifies the client's browser, operating system and other facts. Microsoft provides a specific object that analyzes these strings.
Property	TotalBytes	Provides the total number of bytes the client sent in the body of the request.
Method	BinaryRead	Provides the exact data the client sent to the server as part of a POST request.

Table 4. Request Object API.

2. Response Object

This is the most complex built-in ASP object. It transmits an ASP page's output to the Web client who submitted the request. Specifically, this object is used to add and alter HTTP headers, build page bodies dynamically, and redirect clients to alternative pages automatically. It has many features but not all of them are frequently used. I will present the most important:

Category	**Name**	**Description**
Collections	Cookies	Contains any cookie values received from the client, or that you want to sent to the client.
Collections	Buffer	Indicates whether to buffer page output. Although buffering requires extra resources from the Web server, experience has shown that buffering outgoing ASP pages actually consumes fewer resources than not buffering them. Buffering is the default in Windows 2000.
Collections	ContentType	Specifies the outgoing HTTP content type. Changing this property tells the browser you are sending something other than HTML (XML, plain text, etc).
Collections	Expires	The number of minutes a browser can cache this page.

Category		
Collections	Expires-Absolute	The date and time the browser must discard any cached copies of this page.
Collections	Status	The value of the HTTP status code returned by the Web server.
Methods	AddHeader	Adds or overrides an outgoing HTML header.
Methods	Flush	Immediately sends any buffered output to the Web client.
Methods	Redirect	Sends a message to the browser that tells it to connect to a different URL.
Methods	Write	Writes an expression to the current HTTP output as character data.

Table 5. Response Object API

3. Server Object

This object contains basic properties and methods that are used in almost every ASP page, like:

Category	Name	Description
Property	ScriptTimeout	The amount of time a script can run before timing out.
Methods	CreateObject	Creates an instance of any non-built-in server component the ASP page requires.
Methods	HTMLEncode	Applies HTML encoding to a string.
Methods	MapPath	Maps a virtual path into a physical path. The virtual path can be an absolute path on the current server or a path relative to the current page.
Methods	URLEncode	Applies URL encoding rules, including escape characters, to a string.
Methods	Transfer	Transfers page execution from one page to the next. This method is more efficient than Response.Redirect method.
Methods	GetLastError	Returns a reference to an ASPError object.

Table 6. Server Object API.

4. Application Object

The Application object provides an area where all the pages of the application can exchange data among themselves and a means by which a developer can schedule code to run whenever the application starts or stops. It includes:

Category	Name	Description
Collections	Contents	Contains all items that script commands have added to the Application object.
Collections	StaticObjects	Contains all objects that <object> tags have added to the application.
Methods	Lock	Prevents other clients from modifying Application object properties.
Methods	UnLock	Allow other clients to modify Application object properties.
Methods	Remove	Deletes an item from the Contents collection.
Methods	RemoveAll	Clears the entire collection from memory.
Events	Application_ OnStart	If a subroutine with this name exists in the global.asa file, it will run whenever the application starts.
Events	Application_ OnEnd	If a subroutine with this name exists in the global.asa file, it will run whenever the application terminates.

Table 7. Application Object API.

5. Session Object

The first time a Web client enters an application (requests one of its ASP pages) the Web server creates a Session object for that client. These objects are persistent, meaning that they remain in existence until there is no activity from the Web client for a specified interval (usually 20 minutes). As long as the object exists, it provides a storage area for data passed from one page execution to the next, or from one page to another. It includes:

Category	Name	Description
Collections	Contents	Contains all items that script commands have added to the Session object.
Collections	StaticObjects	Contains all objects that <*object*> tag have added to the session with session scope.

Category		
Methods	Abandon	Destroys a Session object and releases its resources.
Methods	Remove	Deletes an item from the Contents collection.
Methods	RemoveAll	Clears the entire collection from memory.
Events	Session_ OnStart	If a subroutine with this name exists in the global.asa file, it will run whenever the session starts.
Events	Session_ OnEnd	If a subroutine with this name exists in the global.asa file, it will run whenever the session terminates.
Properties	Codepage	The code page (character set) used for symbol mapping.
Properties	LCID	The locale indentifier.
Properties	SessionID	The session identification for this user.
Properties	Timeout	The timeout period for the sessions within the current application, stated in minutes.

Table 8. Session Object API.

6. ObjectContext Object

This object either commits or aborts transactions managed by Microsoft Transaction Server(MTS) initiated by an ASP script. When an ASP page contains the *@TRANSACTION* directive, the page runs in a transaction and does not finish processing until the transaction either succeeds completely or fails. It contains:

Category	Name	Description
Method	SetComplete	Declares that the script is not aware of a reason for the transaction not to complete
Method	SetAbort	Aborts a transaction initiated by an ASP page
Event	OnTransaction Commit	Occurs after a transacted script's transaction commits
Event	OnTransaction Abort	Occurs if the transaction is aborted

Table 9. ObjectContext Object API.

7. ASPError Object

This object increases the ASP error-handling capability letting the developer trapping errors in a custom error message ASP file. The ASPError object is returned by the Server.GetLastError method. It contains:

Category	Name	Description
Property	ASPCode	Returns an error code generated by IIS.
Property	Number	Returns the standard Microsoft's Component Object Model (COM) error code.
Property	Source	Returns the actual source code of the line that caused the error.
Property	FileName	Indicates the name of the ASP file that was being processed when the error occurred.
Property	LineNumber	Indicates the line within the ASP file that generated the error.
Property	Description	Returns a short description of the error.
Property	ASP Description	Returns a more detailed description of the error if it is an ASP-related error.

Table 10. ASPError Object API.

B. JSP IMPLICIT OBJECTS

The JSP container exposes nine internal objects to the JSP developer. They are referred to as implicit objects, because they don't have to be declared or created by the developer in order to be used. These objects will be automatically assigned to specific variable names in the page's scripting language.

Besides the implicit objects, JSP pages, via scripting elements, have all the power for creating, modifying, and interacting with Java objects in order to generate dynamic content. Application-specific classes can be instantiated and values from method calls can be inserted into JSP output. Network resources, such as databases, can be accessed to store and retrieve data for use by JSP pages.

JSP pages are based on the Java servlets technology. Behind the scenes the JSP container automatically creates, compiles, loads, and runs a special servlet to generate the JSP's output, as shown in Figure 3.

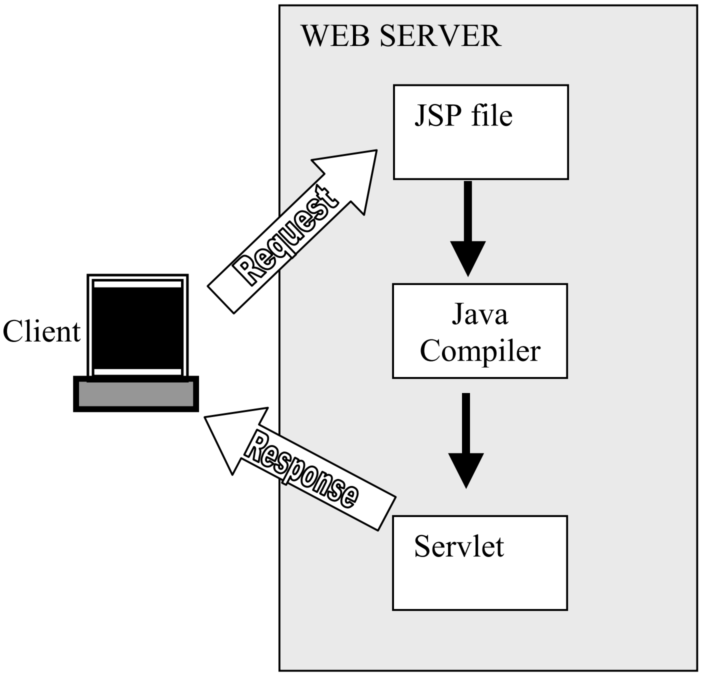


Figure 3. JSP Conversion to Servlet

The JSP implicit objects fall into four major categories: objects related to a JSP page's servlet, objects concerned with page input and output, objects providing information about the context within which a JSP page is being processed, and objects resulting from errors [Ref. 9].

Beyond this functional categorization, four of the JSP implicit objects –request, session, application, and pageContext– have something else in common: the ability to store and retrieve arbitrary attribute values, using the four methods in the table below. By setting and getting attribute values, these objects are able to transfer information between and among JSP pages and servlets as a simple data sharing mechanism. The common methods for attribute management of these four objects are:

Method	Description
setAttribute(key, value)	Associates an attribute value with a key.
getAttributeNames()	Retrieves the names of all attributes associated with the session.
getAttribute(key)	Retrieves the attribute value associated with the key.
removeAttribute(key)	Removes the attribute value associated with the key.

Table 11. JSP Common Methods for Attribute Management.

Lets see each JSP implicit object in detail:

1. Servlet-related Objects

a. Page Object

The *page* object represents the JSP page itself or more specifically, an instance of the servlet class into which the page has been translated. As such, it may be used to call any of the methods defined by that servlet class. Essentially, when the scripting language is Java, the page object is an alias for the *this* variable. This object is rarely used, because the JSP author has direct access to all methods of the JSP's generated servlet. The servlet class is required to implement the javax.servlet.jsp.JspPage interface, and in the case of the HTTP protocol, the javax.servlet.jsp.HttpJspPage interface.

b. Config Object

The *config* object stores servlet configuration data (in the form of initialization parameters) for the servlet, into which a JSP page is compiled. The values for initialization parameters are specified via the deployment descriptor file (web.xml) of the web application. The initialization process can be done using the jspInit() method. This object is an instance of the javax.servlet.ServletConfig interface and has the following methods:

Method	**Description**
getInitParameter(name)	Returns the value of the specified servlet initialization parameter, or null if the named parameter does not exist.
getInitParameterNames()	Returns a list of the names of all initialization parameters for this servlet

Method	
	for this servlet.
getServletContext()	Returns a reference to the servlet context (same as the application implicit variable).
getServletName()	Returns the name of the generated servlet .

Table 12. Config Object API.

2. Input-Output Objects

a. Request Object

The *request* object represents the request that triggered the processing of the current page. For HTTP requests, this object provides access to all of the information associated with a request, including its source, the requested URL, and any headers, cookies, or parameters associated with the request. The request object is required to implement the javax.servlet.ServletRequest interface, and when the protocol is HTTP, it must implement a subclass of this interface, javax.servlet.http.HttpServletRequest. This object provides methods for retrieving request parameters, HTTP headers and miscellaneous functionality such as access to the requested URL and the session. The key methods are:

Method	Description
getParameter(name)	Returns the first value of a single request parameter.
getParameterNames()	Returns the names of all request parameters.
getParameterValues(name)	Retrieves all of the values for a single request parameter.
getHeaderNames()	Retrieves the names of all of the headers associated with the request.
getHeader(name)	Returns the value of a single requested header, as a string.
getHeaders(name)	Returns all of the values for a single requested header.
getIntHeader(name)	Returns the value of a single requested header, as an integer.
getDateHeader(name)	Returns the value of a single requested header, as a date.
getCookies()	Retrieves all of the cookies associated with the request.
getMethod()	Returns the HTTP(GET, POST, etc) method for the request.

Method	
getRequestURI()	Returns the requested URL, up to but not including any query string.
getQueryString()	Returns the query string that follows the URL, if any.
getSession(flag)	Retrieves the session data for the request (the session implicit object), optionally creating it if it doesn't already exist.
getRequestDispatcher(path)	Creates a request dispatcher for the indicated local URL.
getRemoteHost()	Returns the fully qualified name of the host that sent the request.
GetRemoteAddr()	Returns the network address of the host that sent the request.
getRemoteHost()	Returns the name of the user that sent the request, if known.

Table 13. Request Object API.

b. Response Object

The response object represents the response that will be sent back to the user as a result of processing the JSP page. This object implements the javax.servlet.ServletResponse interface and in the case of the HTTP protocol the javax.servlet.http.HttpServletResponse interface. This object provides methods for specifying the content type and encoding of a response, for setting response headers and codes and for URL rewriting. The key methods of this object are:

Method	Description
setContentType()	Set the MIME type and, optionally the character encoding of the response's contents.
getCharacterEncoding()	Returns the character encoding style set for the responses contents.
addCookie(cookie)	Adds the specified cookie to the response.
containsHeader(name)	Checks whether the response includes the named header.
setHeader(name, value)	Assigns the specified string value to the named header.
SetIntHeader(name, value)	Assigns the specified integer value to the named header.
setDateHeader(name, value)	Assigns the specified date value to the named header.
addHeader(name, value)	Assigns the specified string value as a value for the named

Method	
	header.
AddIntHeader(name, value)	Assigns the specified integer value as a value to the named header.
addDateHeader(name, value)	Assigns the specified date value as a value for the named header.
setStatus(code)	Sets the status code for the response (for non-error circumstances).
sendError(status, msg)	Sets the status code and error message for the response.
SendRedirect(url)	Sends a response to the browser indicating it should request an alternate (absolute) URL.
encodeRedirectURL(url)	Encodes a URL for use with the sendRediect() method to include session information.
encodeURL(name)	Encodes a URL used in a link to include session information.

Table 14. Response Object API.

c. Out Object

The *out* object represents the output stream for the page, the contents of which will be sent to the browser as the body of its response. It is an instance of the javax.servlet.jsp.JspWriter class, which extends the standard java.io.Writer class. It provides all the write(), print(), and println() methods for output generation. By taking advantage of this implicit object, output can be generated from within the body of a scriplet without having to temporarily close the scriplet to insert static page content or JSP expressions. In addition the *out* object provides methods for controlling the output buffer and managing its relationship with the output stream that ultimately sends content back to the browser.

Method	Description
isAutoFlush()	Returns true, if the output buffer is automatically flushed when it becomes full, false if an exception is thrown.
getBufferSize()	Returns the size (in bytes) of the output buffer.
getRemaining()	Returns the size (in bytes) of the unused portion of the output buffer.

Method	
clearBuffer()	Clears the contents of the output buffer, discarding them.
clear()	Clears the contents of the output buffer, signaling an error if the buffer has previously been flushed.
newLine()	Writes a (platform specific) line separator to the output buffer.
flush()	Flushes the output buffer, then flushes the output stream.
close()	Closes the output stream, flushing any contents.

Table 15. Out Object API.

3. Contextual Objects

a. Session Object

This JSP implicit object represents an individual user's current session. All of the requests made by a user that are part of a single series of interactions with the web server are considered to be part of a session. If a certain length of time passes (usually 30 minutes) without any new requests from the user, the session expires. The session object provides methods for storing information about the session. Some useful methods are:

Method	Description
getId()	Returns the session id.
getCreationTime()	Returns the time at which the session was created.
getLastAccessedTime()	Returns the last time a request associated with the session was received.
getMaxInactiveInterval()	Returns the maximum time (in seconds) between requests for which the session will be maintained.
setMaxInactiveInterval()	Sets the maximum time (in seconds) between requests for which the session will be maintained.
isNew()	Returns true if user's browser has not yet confirmed the session id.
invalidate()	Discards the session, releasing any objects stored as attributes.

Table 16. Session Object API.

b. Application Object

This implicit object represents the application to which the JSP page belongs. It is an instance of the javax.servlet.ServletContext interface. The application object provides methods for retrieving version information from the servlet container, for accessing server-side resources represented as filenames and URLs, for logging, for setting and getting attribute values, and for accessing initialization parameters associated with the application as a whole. Some useful methods are:

Method	Description
getServerInfo()	Returns the name and version of the servlet container.
getMimeType(filename)	Returns the MIME type for the indicated file.
getResource(path)	Translates a string specifying a URL into an object that accesses the URLs contents either locally or over the network.
getResourceAsStream(path)	Translates a string specifying a URL into an input stream for reading its contents.
getRealPath(path)	Translates a local URL into a pathname in the local file system.
getContext(path)	Returns the application context for the specified local URL.
getRequestDispatcher(path)	Creates a request dispatcher for the indicated local URL.
log(message)	Writes the message to the log file.
log(message,exception)	Writes the message to the log file, along with the stack trace for the specified exception.

Table 17. Application Object API.

c. PageContext Object

This object provides access to all other implicit objects. In addition the pageContext object implements methods for transferring control from the current page to another page, either temporarily to generate output to be included in the output of the current page, or permanently to transfer control altogether. This object is an instance of the javax.servlet.jsp.Page-Context class. It provides methods for programmatically accessing all of the other JSP implicit objects, for dispatching of request from one page to another, and for managing its attributes.

Method	Description
getPage()	Returns the servlet instance for the current page (page object).
getRequest()	Returns the request that initiated the processing of the page.
getResponse()	Returns the response of the page.
getOut()	Returns the current output stream of the page.
getSession()	Returns the session associated with the current page request.
getServletConfig()	Returns the servlet configuration object.
getServletContext()	Returns the context in which the page's servlet runs.
GetException	For error pages, returns the exception passed to the page.
forward(path)	Forwards processing to another URL.
include(path)	Includes the output from processing another local URL.
setAttribute(key,value,scope)	Associates an attribute value with a key in a specific scope.
getAttributeNamesInScope (scope)	Retrieves the names of all attributes in a specific scope.
getAttribute(key,scope)	Retrieves the attribute value associated with a key in a specific scope.
removeAttribute(key,scope)	Removes the attribute value associated with a key in a specific scope.
FindAttribute(name)	Searches all scopes for the named attribute.
getAttributesScope(name)	Returns the scope in which the named attribute is stored.

Table 18. PageContext Object API.

4. Error Handling Objects

Exception Object

This implicit object is not automatically available on every JSP page. Instead, the exception object is only available on pages that have been designated as error pages using the isErrorPage attribute of the page directive. On those JSP pages that are error pages, the exception object will be an instance of the java.lang.Throwable class corresponding to the uncaught error that caused control to be transferred to the error page. Some useful methods are:

Method	Description
getMessage()	Returns the descriptive error message associated with the exception.
printStackTrace(out)	Prints the execution stack in effect when the exception was thrown to the designated output stream.
toString()	Returns a string combining the class name of the exception with its error message (if any).

Table 19. Exception Object API.

C. COMPARISON BETWEEN THE ASP AND JSP OBJECTS

Now, we will attempt a direct comparison between the ASP and JSP objects and their functionality.

1. Request Object API

Feature	ASP Request object	JSP Request object	REMARKS
Retrieving client certificate in X.509 standard (when it is requested from the server).	The ClientCertificate collection contains any submitted client certificate. Additionally, the ServerVariables collection provides extra details for that certificate.	The client certificate can be obtained via the request attribute javax.servlet.request. X509Certificate, which returns a java.security. cert.X509Certificate object.	Both technologies handle certificates efficiently, providing easy access to every field of the submitted client certificate.
Retrieving cookies.	The Cookies collection provides -by name- any submitted cookies by the client.	Method getCookies() returns an array of Cookie objects submitted by the client.	Similar functionality for both.
Retrieving form data (parameters).	Form and QueryString collections are used when the form's *method* is POST or GET respectively. The parameters are	Methods getParameterNames(), getParameter(name) and getParameterValues(name) are used no matter what form *method* was used.	Generally, the functionality is similar. However the ASP implementation is more restrictive because you have to use specific collection based in the

Feature	ASP Request	JSP Request object	REMARKS
	retrieved by name.		form's action method. If no collection is defined then the ASP processor will search all the collections of the Request object, resulting in more overhead.
Retrieving HTTP headers, and details about the client, the client's browser and the Web server.	ServerVariables collection provides all that information, depending on the supplied variable. Also, Microsoft Web servers include a specific object for retrieving client's browser information.	Methods getHeaderNames() and getHeader(name) provide access to the HTTP headers. Also, numerous methods from the Request implicit object provide all the necessary information (via the javax.servlet.http. HttpServletRequest interface)	Both approaches provide a rich variety of methods and variables for retrieving all that information.

Table 20. Request API Comparison.

2. Response Object API

Feature	ASP Response object	JSP Response object	REMARKS
Setting response cookies.	The Cookies collection sets any response cookies.	Method addCookie(cookie) adds any new response cookies.	Similar functionality for both.
Setting response headers.	Method AddHeader and collections like CacheControl, ContentType along with some others provide response header management.	Methods like setHeader(name,value), addHeader(name,value) and containsHeader(name) manage any response headers.	Although the functionality is similar for both, JSP implementation is richer because it can check if a given header has already been set.
Setting the response status codes.	The Status property along with the Redirect method are used for setting status codes.	Methods setStatus(code), sendError(status, msg), and sendRedirect(url) provide status code management.	Similar functionality for both.

Table 21. Response API Comparison.

JSP Response implicit object provides two more methods (**encodeURL(name)** and **encodeRedirectURL(url)**) that support URL rewriting, which is one of the techniques supported by JSP for session management. There are no similar methods in ASP implementation.

3. Session Object API

Feature	ASP Session object	JSP Session object	REMARKS
Setting and retrieving items or objects to the Session object.	Collections Contents and StaticObjects can be used to set or retrieve any items or objects respectively, that are included in the Session object. Also, methods like Remove and RemoveAll remove items from the Session object.	Methods setAttribute(key,value), getAttribute(key), getAttributeNames(), and removeAttribute(key) are used to set, get or remove objects to the Session implicit object.	Basically, the functionality is similar for both.
Managing the Session object.	There are properties for setting the code page, locale identifier and Session timeout. Events like Session_OnStart and Session_OnEnd used when the Session object is created or ended respectively, while method Abandon destroys the current Session object.	There are methods for setting the timeout time, for getting the last accessed time or the creation time, and for checking if the Session object was confirmed by the client's browser. Also method invalidate() discards the current Session.	Both implementations cover the Session management adequately. However, ASP approach has the Session_OnStart and Session_OnEnd events, which provide additional flexibility in initializing and terminating procedures.

Table 22. Session API Comparison.

An important aspect is that ASP maintains Session object using cookies. If the Web client's browser doesn't support cookies or has disabled them, then the Session object cannot be implemented. JSP solves that problem by using URL rewriting (via built-in methods **encodeURL(name)** and **encodeRedirectURL(url)**), when cookies are not available. This is a much more efficient approach.

4. Application Object API

Feature	ASP Application object	JSP Application object	REMARKS
Setting and retrieving items or objects to the Application object.	Collections Contents and StaticObjects can be used to set or retrieve any items or objects respectively, that included in the Application object. Also methods like Remove and RemoveAll remove items from the Application object.	Methods setAttribute(key,value), getAttribute(key), getAttributeNames(), and removeAttribute(key) are used to set, get or remove objects to the Application implicit object.	Similar functionality for both.
Synchronization issues (Application is a common share object) .	Methods Lock and Unlock are provided to modify Application object safely.	The keyword synchronized may be used to provide thread-safe access to the Application object.	Similar functionality for both.
Application object management.	Events like Application_OnStart and Application_OnEnd provide initialization and termination handling.	Methods getInitParameter(name) and getInitParameterNames() provide initialization parameters to the Application object.	While initialization procedures work similarly for both implementations, ASP has an additional utility for termination procedures.

Table 23. Application API Comparison.

Beyond the above comparison, JSP Application implicit object provides methods for retrieving information about servlet container, for accessing server-side resources and also support for logging. ASP provides similar functionality through other objects (e.g. ServerVariables collection of Request object), while logging is completely configurable through IIS Web server environment.

5. Error Object API

Feature	ASP *ASPError* object	JSP Exception object	REMARKS
Information returned by the error object.	It returns the IIS error code, the COM error code, the source of the error, the file and the line that generated the error, a short description, and a detailed description if the error was an ASP-based error.	It returns the descriptive error message, the exception class name, and the execution stack.	Basically, both objects provide similar functionality, returning adequate information about the resulting error.

Table 24. Error API Comparison.

The rest of the ASP and JSP objects cannot be directly compared.

The last of the ASP object is the Server object, which supports some useful methods like HTMLEncode, MapPath, URLEncode, Transfer, GetLastError and the ScriptTimeout property. JSP provides methods like getRealPath(path), the encode method of the URLEncoder class, the sendRedirect(url), and the exception implicit object which are the equivalent of MapPath, URLEncode, Transfer, and GetLastError methods respectively. However, there is no JSP scriptTimeout property, or HTML encoding method (although it can be programmed easily).

JSP implicit objects PageContext, Out, Config and Page support additional functionality like access to every implicit object, output generation, initialization procedures, and an alias to the current servlet instance, respectively. However, ASP through the objects that we have already seen, provide similar functionality to those four JSP objects.

Concluding, we can say that generally the built-in objects of both implementations offer similar functionality with minor exceptions - richer HTTP management from JSP, more flexibility with initialization and termination procedures in Application and Session objects from ASP, etc. However, we have to mention the inadequate session management of the ASP technology (relies only on cookies), while on the other side JSP technology offers an efficient way of session management, supporting URL rewriting beyond cookies.

IV. ASP-JSP Application Architecture

A. ANALYSIS

When designing web applications of any complexity using technologies like ASP or JSP, it is very helpful to have the high-level application architecture separated in discrete areas or layers (Figure 4) like:

- The **presentation** layer, which is the part of the application that the user can sees or interacts directly
- The **control** layer, which controls the overall information flow between the application and the user
- The **application logic** layer, which includes all the information processing, and communication with back-end resources like databases

This architectural separation helps both developers and designers of the web application. It helps to designate borders between each team's responsibility area, facilitates the abstraction of the application's modules, future modifications and maintenance are simplified, and the application's lifecycle is usually longer. Each one of the above three layers can be implemented in discrete components or they can be combined all in one. The choice is up to a number of parameters like:

- The size and the complexity of the application
- The available time
- The number of the available designers and developers and their skill

However, the less the discrete components will be implemented, the less modularity and high-level abstraction will be achieved.

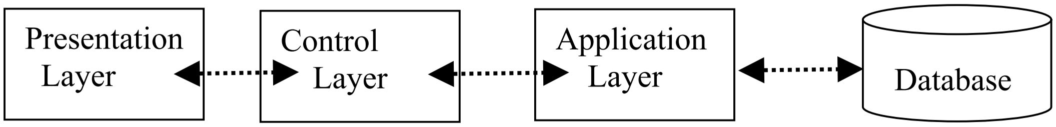


Figure 4. Architecture Layers.

Lets see each layer in more detail:

1. Presentation Layer

This layer consists mainly of display elements like HTML or XML. It is the user interface of the application receiving input from or displaying results to the user. This layer doesn’t care how a user’s request was processed by the application. It just receives the request and submits it in a background component or module. Additionally, it doesn’t care from what component or module the application’s response was initiated. It just receives it and presents it to the user, formatting it appropriately.

2. Application Layer

This layer is the core part of the application. It implements all the necessary business logic executing queries to a database, making complex calculations and manipulating data. Its overall operation is independent of any data presentation. It doesn’t need to know anything about HTML and if designed properly it can be reused in other web-based or non web-based applications.

3. Control Layer

This layer stands between the presentation and the application layers. It receives requests from the presentation layer making decisions about where it will deliver them. If for example, the request is about a database update, it delivers the request data only to the component or module that does exactly that operation. Also the control layer decides if a request is legitimate or not, taking an appropriate action. Finally it receives from the application logic layer any completed request and delivers it to the user via the presentation layer. Generally, this layer decides ***which*** application module will process a request and ***when*** (the ***how*** portion of the process is left to the other two layers).

However, designing the application’s architecture in layers is not an easy task. Mainly, there are two approaches to do that:

- The page-centric approach
- The Model-View-Control (MVC) approach

4. Page-centric Approach

In the page-centric approach, the application consists of interrelated server pages that include all three layers (Figure 5). Every page must perform, to some extent, tasks from each layer. For example every page receives directly or indirectly user input, makes the necessary manipulation, presents some results to the user and finally awaits a user's response in order to forward control to the next page. Of course this doesn't mean that the three layers don't exist. They certainly exist, but to be more precise, they co-exist in every page, blurring the distinction between them.

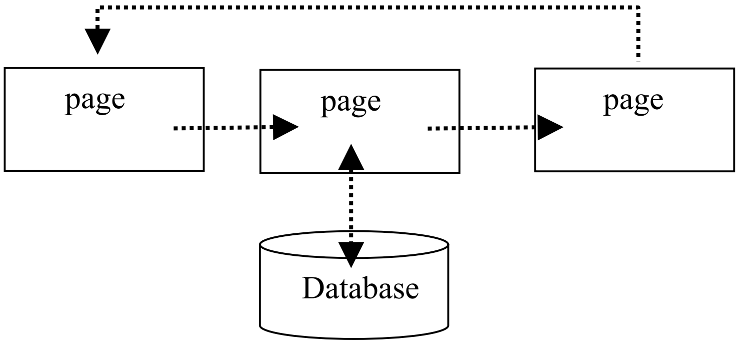


Figure 5. Page-centric Approach.

Nevertheless, the page-centric approach is relatively simple from the architectural perspective. It is appropriate for small applications, with little abstraction. It can be used when we need immediate results, there is not enough time, by individuals or very small developing teams, and for developing prototypes. However, there are some serious problems with this approach such as:

a. Maintainability

Because every page includes presentation, logic, and control code, without discrete borders between them, it usually requires a high degree of interaction between the page designers and the developers whenever any modifications need to be done.

b. Flow Control

Because of the lack of central control, each page must maintain its own protective code to prohibit out of order execution, invalid request parameters or any other unexpected behavior. This is feasible for a few pages, but a real headache for bigger applications.

5. MVC Approach

In this approach all the server pages are used mainly for presentation, leaving the application flow control and the application logic to intermediate discrete component(s) (Figure 6). All requests are routed from the front-end (server pages) to the controlling component(s), which perform all the necessary checks before they are forwarded to component(s) that incorporate the application's logic and data structures. When all the requests are processed they are returned back to the server pages again, for presentation. More specifically the intermediate components can do:

- Application flow control between the presentation layer and the application logic.
- Perform an action (database query, data manipulation, etc) based on the submitted request from the server pages.
- Deliver requests back to the server pages for display.

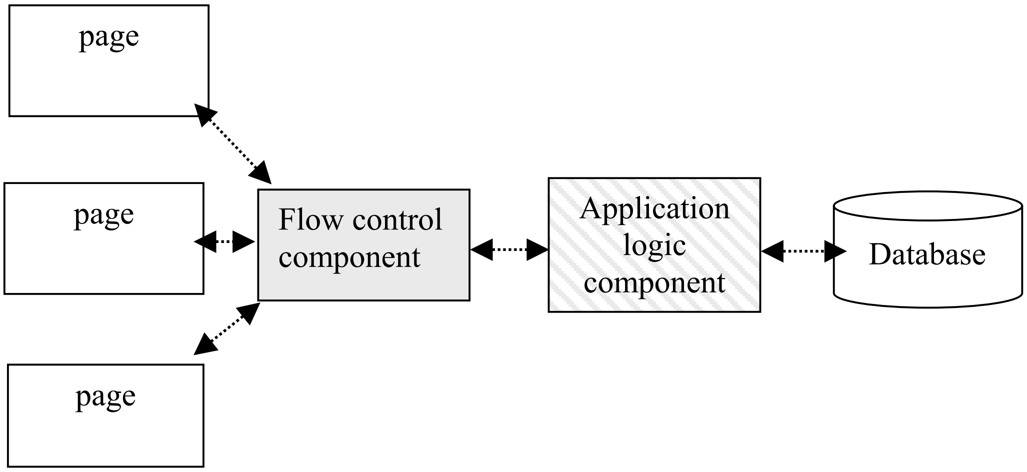


Figure 6. MVC Approach.

In this approach the server pages are relatively simple because their only task is presentation, without any application logic. This makes maintenance easier, because only the web-designers are needed in case of any modifications, not the developers. On the other side, intermediate components include only application logic and flow of control, having no presentation tasks. This also facilitates maintenance because only developers are needed in order to perform any modifications in these components.

As a result of this approach, we have loosely coupled server pages, because they don’t have to be directly aware of any other page inside the application. The abstraction is also increased between the presentation and the application logic. Last but not least, both the server pages and the intermediate components can be reused in other applications, with little or no modifications, saving valuable time.

B. COMPARISON

ASP and JSP technologies were introduced in order to provide a simplified, fast way to create web pages that display dynamically generated content. They both succeed in that task. Also, they are both able to follow the above two architectural approaches, but not in the same way and scale, which actually makes the difference between these two technologies from the architectural point of view. But first lets examine each one separately:

1. ASP Architecture

ASP is able to follow both the page-centric and the MVC approaches. However, there are some serious limitations in the second approach.

In case of the page-centric approach, which is the most popular and common implementation of the ASP technology, the application consists only of ASP pages, html pages, and possibly a database. All application control and logic, along with the data presentation, are included in the ASP pages. This is very convenient when the application is not too large or the logic to be implemented not too complicated. If one (or both) of these occurs then the ASP code can easily became very complex, introducing serious maintenance problems. However, the page-centric approach with ASP can provide speed during development phase and relatively quick results, because of Microsoft's built-in support of a number of useful and handy components, like ActiveX Data Objects (ADO).

In the case of the MVC approach, ASP provides to the developer the ability to use the Component Object Model (COM). Which means that the developer can create, using a compiled language like C++ or Visual Basic, COM objects encapsulating part of the application logic. However, this is not an easy task. It involves a lot of complexity and requires very skilled developers. That's why that approach is not as common as the page-centric approach in the ASP world. Last but not least, COM objects, although they can incorporate application's logic and control, cannot totally free-up ASP pages from including application logic, allowing them to perform only presentation tasks.

2. JSP Architecture

Applications that are built in with JSP technology, can also adopt both architectural approaches. They may consist of JSP and HTML pages like the ASP approach or they can incorporate the MVC (servlet-centric) approach. In the first case we have all the advantages and limitations already mentioned in the previous paragraphs. In the second case things are different.

Beyond JSP pages, the Java platform provides a number of web components, like servlets, javabeans, enterprise javabeans or even more, regular Java classes, which can be incorporated in a JSP web-application. Specifically, each of the above Java objects can be used in the following manner:

- The **servlets** as a flow control object or to include portion of the application logic. However, including large portion of the application logic in servlets is not recommended in large-scale applications.
- The **javabeans**, including part of the application logic and facilitating the presentation tasks of the JSP pages.
- The **enterprise javabeans** (**EJBs**), in case of large, complex and distributed applications, encapsulating business logic into reusable server-side components.
- The **regular Java classes** as background objects that have specific tasks like interaction with databases.

Figure 7 shows all the interrelated components of a JSP web application using the MVC approach.

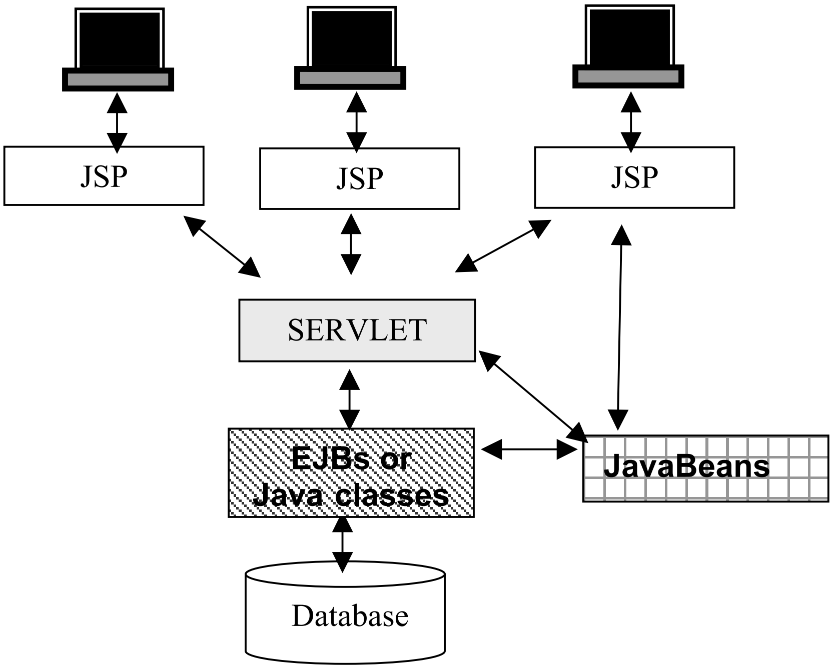


Figure 7. MVC Approach with JSP.

Normally, the role of the flow control in a web application is assigned to servlets because they can easily incorporate any portion of Java code needed for that task. Being freed from any presentation task, servlets can cleanly handle any portion of JSP requests, forwarding them to the appropriate application logic component for processing and returning responses to the JSP pages, which only task is presentation.

Data representation can be easily implemented in javabeans objects, which are created from application logic components and sent via servlets to the JSP pages for presentation.

Application logic can be encapsulated into EJBs, javabeans, or regular java classes that work in the background, receiving requests from JSPs, via servlets, making all the necessary processing and sending back to a JSP any generated response, again via servlets.

Having an application incorporating such a variety of objects, it is easy to create dedicated components for each architecture layer (presentation, control, and application logic). This greatly facilitates maintenance, flow control, distributed application development, and reuse of components. However, usually this approach needs more designing and development time, mainly at start-up, and needs more skilled developers and designers. Of course, the pay-off comes later, during the lifecycle of the web application.

3. Conclusions

In my ASP and JSP prototype applications, I choose the page-centric model for the ASP and the MVC (servlet-centric) model for the JSP. I consider the MVC model more challenging than the page-centric model. Although I have a better background in Java technology generally than in ASP programming, it took me more time to implement the JSP application, than the ASP. The reason was that building an application with the MVC model needs much more time in the design phase and many more components. However, when the basic application is up and running, making modifications or adding more functionality is an easy thing. I found out that maintenance was facilitated from the servlet-centric model because the three layers (presentation, control, application logic) were well separated, and the application had many components, each of which had their own unique functionality. When something needed to be fixed, the only thing required was to find the appropriate component and make the appropriate correction, without worrying about the other parts of the application. Reuse of code was also an important issue. In my JSP application I have a component (Java class) that handles all the database interaction. This component is very loosely coupled with the other two parts of the application (servlets and JSPs), and it is database independent. In fact it can be used with any other type of Java application that needs database interaction.

On the other side, ASP application is consist of ASP pages only. The initial implementation effort was less than with JSP, and the results were achieved faster. However, each page implementation includes presentation and control tasks, incorporating several lines of application logic. This makes the distinction between the different application layers not as clear as with JSP. This had its impact with maintenance, because in case of any modification I had to go through many more lines of code, in more than one component.

I could argue that, from an architectural point of view, JSP technology is more attractive because it offers both architectural approaches in a more complete and flexible way. It is easier to develop servlets and javabeans for the JSP servlet-centric approach than COM components in the ASP. Also, while using the page-centric approach with ASPs is faster than with JSPs, mainly because of the ease of use of Microsoft’s components, like ADO, and the built-in support of the ASPs in Windows, I believe that the JSP’s servlet-centric approach is a complete solution for every type and size of web application, following the modern trend of the object-oriented approach.

THIS PAGE INTENTIONALLY LEFT BLANK

V. ASP - JSP Performance

A. ANALYSIS

Tuning application performance in server side programming like Active Server Pages(ASP) or Java Server Pages(JSP) is a significant challenge. There are many different areas that may affect the performance of an ASP or JSP web-application:

- The ASP and JSP pages themselves
- The database
- The web-server
- The HTML pages
- The network resources
- The HTML client (browsers)

We will examine the first three, due to their direct relation to the server-side programming, and great impact on the overall performance of the web application. I will present the specific, performance-related factors and how they affect ASP and JSP pages. Additionally, I will present a performance test that I made using my implementation of ASP and JSP technology.

Besides these factors that are specific for each ASP or JSP implementation, there are some built-in features that benefit one technology over another. Specifically, ASP is a dynamic content system that relies on interpreted scripting languages like VBScript or JavaScript. The ASP file containing scripts in those interpreted languages must be parsed each time that file is requested. This parsing incurs extra overhead and delays. JSP doesn't suffer from that kind of parsing. JSP files are parsed only the first time they are requested, when they compiled into servlets. As a result, JSP will be significantly slower than ASP for this first request, because of the compilation step. However, in every subsequent request, the already compiled JSP servlet that remained in server memory will immediately respond without further delays, resulting in better response times than ASP.

Now, lets see which factors affect ASP or JSP performance and how:

Factors	Description
Script blocks	Script blocks inside a page must be combined as much as possible. Each transition between script code and HTML code creates work for the page processor. This affects both technologies equally. It's better to have a few large blocks of script code rather than many small ones. Things can be worse for ASP pages when more than one scripting languages is used, like *VBScript* and *JavaScript*. Then the server has to load more than one script interpreter, which degrades performance. This is not a problem for JSP, because it currently supports only the *Java* language as its scripting language
Preparing strings for output	It's also faster to output a long string than many repeated short ones. So, creating first a long string from a number of small ones and then sending it to the output is always a good idea. This applies equally to both technologies. However, in the case of JSP, concatenating *String* objects does not help performance, because each String concatenation creates additional *String* objects. Instead, the *StringBuffer* object –with its *append()* method– must be used, because it avoids the creation of unnecessary intermediate objects.
Use of Application object for frequently accessed data	If there are pages or other web-related objects that run frequently and use resource-intensive methods to obtain relatively static data, it is possible to improve performance by retrieving that data once and storing it to the *Application* object. For example, if a web site frequently needs to translate the same relative path to its physical path then it will frequently need to call translation methods. In a real site, that may happen thousand times per hour. Instead, doing the translation only once, when the web-application is started, and by storing the result to the *Application* object, we make the translated path directly available to any web page, without any need for further calls to translation methods. This applies equally to both technologies. However, because *Application* object remains constantly in memory, we must use it in a balanced way, avoiding storing data by the megabytes.
Accessing the same data multiple times from the same page	Instead of accessing the same data many times –e.g. inside a loop– from a built-in object (ASP) or an implicit object (JSP), it is preferable to access that data once and then store it to a local variable. Then, calling that variable inside the page will be much faster and more efficient.
Object creation	Object creation consumes resources. It's always better to avoid creating unnecessary objects. More specifically, in the JSP world, most Java Virtual Machines (JVM) are using a global object heap that must be locked for each new memory allocation. While any servlet (the compiled JSP) is creating a new object or allocating additional memory, no other servlet can do so. In a multithreaded world, this results in delays. In the ASP world, it is recommended, for performance reasons, the use of the

Factors	
	<*object*> tag to create objects, instead of using the common *Server* object's *CreateObject* method.
Session object	The *Session* object is created for each user that accesses the web application. *Session* object, like every other object, consumes resources. Avoid creating that object when session management isn't needed. This applies in both ASP and JSP.
Output buffering	Output buffering was proved to improve performance. That's why it is by default enabled in both JSP and ASP (but only in IIS 5.0, not in IIS 4.0). Although disabling buffering is possible in both technologies, it is usually not recommended.
Connection status	Before any lengthy process during the execution of our pages, it is sometimes useful to check if the client is still connected or have abandoned our page. In the second case, further processing is a waste of time. ASP offers the *Response* object's *IsClientConnected* method, which does exactly that. JSP has no similar built-in method.
Timeouts	Timeouts affect performance directly and they can be set for scripts and sessions. Short timeouts may waste resources without any result, while long ones may lock resources waiting for something that will never happen. Consequently, setting timeout is usually application specific. ASP (through IIS) provides the way for setting script timeout for the duration of page execution and session timeout between requests. JSP, through *HttpSession* object or *Tomcat's web.xml* file, provides the way to set session timeout. I didn't find any way to set script timeouts for JSPs.
Browser side scripting	Browser-side scripting reduces the web server's load by moving work to the browser's computer and by reducing the number of requests that the web visitor needs to make. An excellent area of browser-side scripting is the user input validation. However, this type of scripting is useless if a browser doesn't support it, or is disabled. This is applicable to both technologies.
Synchronization	Synchronization is often necessary, especially in multithreaded systems, but slows down response times. Consequently, we have to keep synchronized blocks as small as possible. This is applicable to both technologies. Synchronization issues are more common in multithreaded environments like JSP web applications, which include many synchronized blocks, or objects like *Vector* and *Hashtable*, which are automatically synchronized. It has been shown that adding synchronization can slow the process by a factor of eight (Casey Kochmer).
Concurrent users	The number of concurrent users affects dramatically the performance of the ASP and JSP pages. This is an important factor for large web sites. ASP works well when the concurrent users are less than 500. After that the response is sluggish. On the other side, JSP appears to have a very

Factors	
	well performance even if the number of concurrent users is more than 800 (see related articles in www.jspinsider.com and www.geocities.com/anjali_katariya/index.html)
Choosing JSP container	While ASP has to stick with the IIS web server, JSP has a variety of choices between available JSP containers. *Tomcat*, *Resin*, and *JRun* are some of them, which offer different levels of performance, when serving JSP pages. Also, integration with a well-known web server like *Apache* or *IIS* can further enhance the overall application performance, leaving static web pages to be served from the web server instead of the JSP container.
Database design	Database design is a great performance factor in a web database application. Tuning database usage generally provides greater payback than optimizing the rest of the application code. Query execution can slow down or speed-up significantly the execution time of either ASP or JSP pages. However, keep in mind that strict database *normalization* doesn't always lead to performance gains. We have to adapt, in some extend, the design of our database to the specific needs of our application.
Database indexes	Under certain conditions indexes can speed-up query execution. They must be used for large database tables, which are frequently used and updated.
Store procedures and stored queries	Moving application logic into the database when possible, reduces the load to the web server. Store procedures and stored queries are an excellent way to do so. Databases optimize these stored procedures and queries during creation time and execute them more efficient during run time. Both ASP and JSP pages can invoke store procedures and queries.
Result Set's cursors and lock type	Different types of cursor means different performance. Chose the *forward-only* cursor when there is no need to scroll back and forth, or to keep our recordset always updated, because it is the fastest. Chose the *dynamic* or *sensitive* type of cursor only when keeping our recordset updated is necessary. Lock type also affects performance. *Read-only* option is the most efficient when updates are not needed. Both ASP and JSP support these types of cursors and lock types.
Connection pooling	Opening a connection is a relatively time-consuming process. *Connection pooling* is a technique that keeps a number of connections open and ready to use. The number of connections depends on the needs of the application and the system resources. JSP uses that technique by using Java classes that do just that: initialize a number of active connections to the database and make them available on demand. ASP supports that technique through the built-in Windows *ODBC Data Source* application, which uses a somewhat different technique: It doesn't create initially new connections but retains any new connection for some period of time after an ASP page releases them. JSP appears to use *connection pooling* more efficiently than ASP, making a number of connections

Factors	
	available when the web application is started and being more configurable. The connection pooling tests using my ASP and JSP applications verify this (see results at the end of this chapter).
Storing Connection or ResultSet objects to Session objects	This practice turned out to be bad, because Connection or ResultSet objects remained in memory for the whole session time (the time the visitor spent on the site plus 20-30 minutes until the session expired). As a result, it could consume valuable system resources if the site had many concurrent visitors. Connection pooling was proved to be more effective for both ASP and JSP and should be used instead.
ODBC – JDBC drivers	Drivers affect dramatically database connections. Response times, cursor and lock types are important features of a driver and must be tested thoroughly before its regular use. This is even more necessary for JSP, due to the various types of drivers (Type 1-4) and their implementations. Additionally, the JDBC-ODBC bridge has known memory leak problems and it is not recommended for production systems.

Table 25. ASP-JSP Performance Factors.

B. COMPARISON AND CONCLUSIONS

We built two web applications, one with the ASP and another with the JSP technology. They provide to the user exactly the same layout, interface, and functionality. The difference is in the underlying application architecture, which is totally different (I explain the architectural details in chapter IV "ASP-JSP ARCHITECTURE") and the different way the two applications handle all the above performance-related factors. The details of that test are as follows:

- The database is the same for both applications. Microsoft Access 2000 was used.
- Connection pooling was tested in both cases.
- I used both simple and complex (nested) queries to display, add, update, delete and navigate through the database records.
- The ODBC-JDBC bridge was used for the JSP connection.
- New JDBC features were used like the *CallableStatement* interface, using stored procedures.
- Connections to the database was tested in two ways:

- Database and web server in the same machine.
- Database in different machine than web server, using LAN connection.

The results of the database access times are shown in the following table (in each case, access times were estimated as the mean value of thirty tries):

Type of connection	**Time to perform the database query in milliseconds**	
	ASP	**JSP**
Database in the same machine with web server with no connection pooling	160.6	110.3
Database in the same machine with web server with connection pooling	150.7	70.5
Database in a different machine with web server (LAN connection) with no connection pooling	281.3	236.67
Database in a different machine with web server (LAN connection) with connection pooling	280.6	91.7

Table 26. Database Access Results.

We observe that:

➢ In every case, JSP connections were much faster than ASP ones.

➢ Connection pooling really improves performance in the JSP case, especially in remote database access, where the benefit is huge.

➢ Connection pooling doesn't offer any significant benefit in the ASP case.

I think we can find the reasons of such performance difference between the two technologies in:

❖ The nature of the JSP technology that keeps the compiled JSP servlet in memory, ready to respond in each request, without parsing each time the JSP file. On the other side, ASP files were parsed with each request from the ASP processor, resulting in additional delays.

❖ The connection pooling in the case of JSP reduced the access time dramatically, while in the case of ASP didn't help much.

We have to mention that the ODBC-JDBC is the weakest and slowest Java driver and also that Tomcat isn't the fastest JSP container. These facts certainly affected the JSP performance that could have been better.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. ASP - JSP Web Security

A. ANALYSIS

We could define Security as the mechanism that keeps sensitive information protected and available only to authorized users. Speaking about the Web, we could relate Security to four important issues:

- **Authentication**, which is the ability to verify the identities of the parties involved.
- **Authorization**, which limits access to resources to a select set of users or programs.
- **Confidentiality**, which ensures that only the parties involved can understand the communication.
- **Integrity**, which ensures that the content of the communication is not changed during transmission.

ASP and JSP web security is mostly related to the underlying web server and operating system. ASP uses Microsoft's Internet Information Services (IIS) as the web server and Windows as the operating system. JSP needs a JSP container to operate. Tomcat for example, which is the JSP reference implementation can be used as either a stand-alone container or as an add-on to an existing web server (like Apache, IIS, or Netscape servers). JSP can run under any operating system. Besides any web server and operating system support, each technology may be used for building custom systems in order to apply client authentication and authorization, as well as client certificate validation. Now, lets see in detail how each of these two technologies implement web security.

1. JSP

JSP implement client authentication using the following techniques:

- Basic
- Digest

- Form
- X.509 client certificate
- Custom

Beginning with Servlet API version 2.2 and JSP API version 1.1, the technique for configuring authentication has been standardized. Now, configuration of security policies can be accomplished in a portable manner using the *web.xml* deployment descriptor. In this section I will describe only the security related tags of the *web.xml*, not the entire file syntax. Specifically, the <*auth-method*> tag, which resides within a <*login-config*> tag, defines the login methodology to be used by this application. It allows BASIC, DIGEST, FORM, and CLIENT-CERT values representing authentication types of Basic, Digest, form-based and client-side certificates, respectively. Figure 8 shows the appropriate syntax of these tags.

```
<login-config>
    <auth-method>
      BASIC     <!-- BASIC, DIGEST, FORM, CLIENT-CERT -->
    </auth-method>
    <realm-name>
      Default   <!-- optional, only useful for Basic or Digest-->
    </realm-name>

</login-config>
```

Figure 8. Login Configuration Tags

The <*realm-name*> tag specifies the login realm to use and it has meaning only for Basic and Digest authentication.

The application's authentication method defined in the <*auth-method*> tag is enforced only for those files (servlets, JSPs or HTML pages) that are mentioned inside a <*security-constaint*> tag. Each such tag restricts one file. We can define as many <*security-constaint*> tags as we want in order to restrict access to our files.

The last part of the authentication chain is to somehow define a list (either in a database or in a simple file) of valid usernames and passwords available to the server. The server will look up that list each time a client submits their credentials, in order to grant access or not. For Tomcat 3.2 you specify users in the *conf/tomcat-users.xml* file. This XML file is unencrypted, with usernames, and passwords in plain text.

Finally, when we have applied all the security restrictions in our *web.xml* file, and a client request occurs for a restricted page, the web server denies access instructing the browser to ask for the user credentials. If the user submits the appropriate credentials, then access is granted (authorization issues occur at this point, but we will discuss those shortly). Otherwise, access is denied and the user is prompted for their credentials again by the browser.

The details of each authentication type is as follows:

a. Basic Authentication

Basic authentication is a simple authentication protocol defined as part of the HTTP 1.0 protocol defined in RFC 2617 (available at http://www.ietf.org/rfc/rfc2617.txt). Although virtually all web servers and web browsers support this protocol, it is very weak because passwords are transmitted over the network, thinly disguised by a well-known and easily reversed Base64 encoding. Anyone monitoring the TSP/IP data stream has full and immediate access to all the information being exchanged, unless there is an additional Secure Socket Layer (SSL) encryption employed. However, basic authentication works well through proxy servers and firewalls.

b. Digest Authentication

Digest authentication is a reasonably new authentication scheme that is part of the HTTP 1.1 protocol defined also in RFC 2617. It is a variation of the Basic authentication. Instead of transmitting a password over the network directly, a digest of the password is used instead. The digest is produced by hashing (using *MD5* encryption algorithm) the username, password, URI, HTTP request method, and a randomly generated value provided by the server. Both sides of the transaction know the password and use it to compute digests. If the digests match, access is granted. Transactions are thus more secure than Basic authentication, because no password is transmitted in plain text. However there are some serious limitations:

- Currently only Microsoft Internet Explorer 5.0 or later supports digest authentication
- Both Digest and Basic authentications are subject to a *replay* attack [Ref. 4, 7]

c. Form-Based Authentication

JSPs can also perform authentication without relying on HTTP authentication, by using HTML forms instead. Using this technique allows users to enter a site through a well-designed, descriptive, and friendly login page. Form-based authentication is built into JSP 1.1 and Servlet 2.2 API. Figure 9 shows how to use this type of authentication with the *web.xml* file.

```
<login-config>
    <auth-method>
      FORM     <!-- BASIC, DIGEST, FORM, CLIENT-CERT -->
    </auth-method>
    <form-login-config>   <!—only useful for FORM -->
          <form-login-page>
    /loginpage.html
          </form-login-page>
          <form-error-page>
                /errorpage.html
          </form-error-page>
    </form-login-config>
 </login-config>
```

Figure 9. Form-Based Authentication Tags.

The <*auth-method*> has been changed to FORM. The <*realm-name*> tag has also been replaced with a <*form-login-config*> tag that specifies the login page and the error page to use for the authentication. Any time the server receives a request for a protected resource, the server checks (via Session object) if the user has already logged in. If they have, access is granted (as long as the user is authorized to access that page). If they haven't logged in, the user is redirected to the login page (loginpage.html), where the user is prompted to enter their credentials. If the submitted credentials are correct then the user is granted access. Otherwise, the server redirects the user to the error page (errorpage.html), indicating the type of error.

Compared with basic or digest authentication, form-based login has the advantage that the user enters the site through a friendly login page. However, it still transmits the user password in plain text.

d. X.509 Client Certificate Authentication

When security matters most, digital certificates are the best solution. SSL 3.0, which provides both server and client authentication using certificates, offers a higher degree of confidentiality, integrity, and authentication. There is no need for user credentials, because each party (server, and client) has to submit their own, signed certificate. If the certificates are valid then the entire connection is encrypted and protected, providing strong authentication, confidentiality, and integrity.

JSP or servlets use the *web.xml* file to require SSL from the server, via the <*user-data-constraint*> tag, which is contained inside the <*security-constraint*> tag. Figure 10 shows the exact syntax.

```
<security-constraint>
    …..
<user-data-constraint>
  <transport-guarantee>
     CONFIDENTIAL  <!- - INTEGRAL or CONFIDENTIAL
  </transport-guarantee>
</user-data-constraint>
</security-constraint>
```

Figure 10. Client Certificate Authentication Tags.

The <*transport-guarentee*> tag has two legal values: INTEGRAL and CONFIDENTIAL. INTEGRAL requires the data must be guaranteed not to change in transit. CONFIDENTIAL requires the data must be guaranteed not to have been read by an unauthorized third party in transit. CONFIDENTIAL implies INTEGRAL, and it is usually the standard guarantee.

After requiring SSL from the server we can ask client authentication based on certificates using the <*login-config*> tag and defining *CLIENT-CERT* inside the <*auth-method*> tag, as we have already seen above.

As with Basic and Digest authentication, all of the SSL details are handled by the server, transparent to JSPs or servlets. However, the JSP or servlet can retrieve the client's certificate as a request attribute:

```
java.security.cert.X509Certificate cert =
        (java.security.cert.X509Certificate)
request.getAttribute("javax.servlet.request.X509Certificate");
```

For any server running on J2SE 1.2 or supporting J2EE 1.2 or later, the *request* attribute *javax.servlet.request.X509Certificate* will return a *java.security.cert.X509Certificate* object representing an X.509v3 certificate. That certificate can be picked apart and checked for validity, issuer serial number, signature, and so forth.

e. Authorization

Before we proceed to the custom authentication and authorization lets see how JSP implements authorization. JSP follows the model of role-based authorization. With this model, access permissions are granted to an abstract entity called a security role, and access is allowed only to users or groups of users who are part of that given role. The deployment descriptor specifies the type of access granted to each role, using the <*security-role*> and <*auth-constraint*> tags, but does not specify that role to user or group mapping. That is done with server specific tools, using database tables, text files, or the operating system. Tomcat 3.2 implements that kind of mapping using the *tomcat-*

users.xml file. In that file, for each user we define a username, password and a role. Figure 11, shows a sample *tomcat-users.xml* file.

```
<tomcat-users>
    <user name="Gil"    password="d45n6rc"   roles="engineer" />
    <user name="Alice" password="id45gce"   roles="engineer, guest" />
   <user name="Bob"   password="3k6u7jm"  roles="manager, guest" />
</tomcat-users>
```

Figure 11. A Sample T*omcat-users.xml* File.

So, using Example 4, if a file is protected with the *manager* role, then only user *Bob* can access that file.

z. Custom Authentication and Authorization

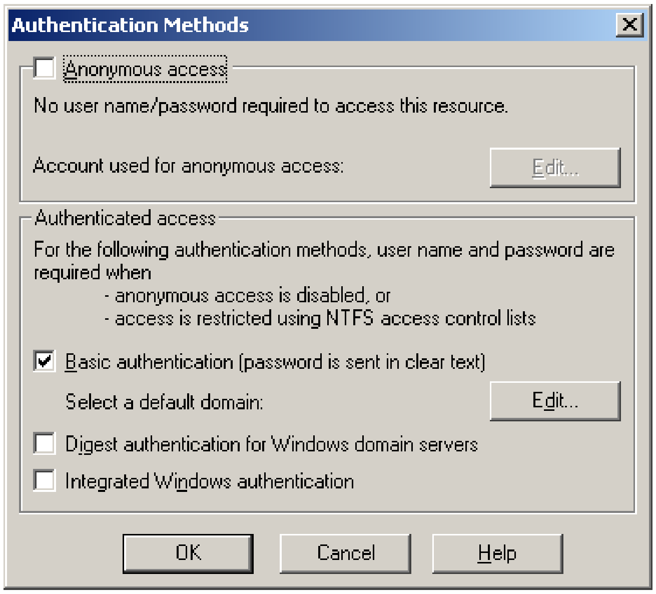
Normally, client authentication and authorization is handled by the web server, using the settings in the deployment descriptor file. However, there are cases where the desired security policy cannot be implemented by the web server. Some applications require from the user more than a username and a password. For example, banking applications may require a username, password, and a PIN. In this case the solution will be to create a custom login system that will handle that job. This can be done using JSPs and servlets. Usually, such a system consists of a login page that receives the user's credentials and sends them to a login controller servlet. This servlet makes the necessary checks between the submitted credential and a database table that keeps the authorized users. If the credentials are valid then access is granted. If the credentials are invalid then the user is redirected again to the login page. In case the user tries to access directly a page, other than the login page, access is denied and they are redirected to the login page. The details of such an implementation can be seen in my JSP-servlet implementation.

2. ASP

ASP security relies totally on the IIS web server and the underlying Windows operating system. Actually IIS is one of the services of the operating system. Windows (NT or 2000) represents each user (principal) with an account, which may belong to one or more user groups. Access or authorization is governed by Access Control Lists (ACL). ACLs associate principals with resources such as files. An ACL contains access control entries (ACEs), and each ACE contains information about what principal can do what to the resource.

Figure 12, shows the IIS handy Graphical User Interface (GUI), which supports the following authentication techniques:

- Anonymous access
- Basic
- Digest
- Integrated Windows
- X.509 client certificate



Authentication Methods

☐ Anonymous access

No user name/password required to access this resource.

Account used for anonymous access:

Authenticated access

For the following authentication methods, user name and password are required when
- anonymous access is disabled, or
- access is restricted using NTFS access control lists

☑ Basic authentication (password is sent in clear text)

Select a default domain:

☐ Digest authentication for Windows domain servers

☐ Integrated Windows authentication

Figure 12. Authentication Methods in IIS 5.0.

Being part of the HTTP protocol, Basic and Digest authentication are implemented from IIS similarly to the JSP. All their capabilities and limitations apply to Windows also. Consequently, we will examine only the rest of the authentication schemes.

a. Anonymous Access

Technically, anonymous access is not an authentication scheme because the calling user is never asked to present credentials. However, because Windows require that all users authenticate themselves before they access any resource, IIS provides a default user account called IUSR_machinename as the Anonymous User account for anonymous access. All anonymous access is performed in the context of this account.

b. Integrated Windows

This type of authentication incorporates two different authentication protocols: the native Windows NT Challenge/Response (NTLM) and the Kerberos V5 authentication protocols. Kerberos is preferred because it is faster and more secure than NTLM. More important, Kerberos is able to authenticate both the server and the client, while NTLM authenticates only the client. Kerberos is a new feature and operates in the Windows 2000 environment only. NTLM is kept for maintaining backward compatibility. However, only Windows machines and the Microsoft Internet Explorer support this type of authentication.

c. X.509 Client Certificate

As we have already mentioned in the JSP section, SSL 3.0 is the best way to check server and client authentication, integrity and confidentiality. IIS offers an easy, graphical way to configure SSL by providing wizards for:

- ➢ Enrollment for a server certificate from a Certificate Authority (CA), such as Verisign, or use the Microsoft Security Services to create your own certificate for use in an Intranet.
- ➢ Certificate Trust List (CTL) creation. This list contains CAs, which their certificates can be trusted from IIS.
- ➢ Mapping client certificates to user accounts, facilitating the authentication and authorizations procedures.

When all these settings are done, IIS can handle all the details of an SSL connection transparently for the user.

Besides that, ASP can retrieve client certificates using *Request.ClientCertificate* collection. *ClientCertificate* collection is a group of client certificate related attributes that belong to the *Request* built-in object of the ASP technology (see Chapter 2 "ASP-JSP objects" for more details). This collection contains attributes describing the certificate's issuer, user, serial number, validity period and much more. All that information is available to ASP, which can easily validate a client certificate.

d. Authorization

As stated before, IIS performs authorization using Windows ACLs. Besides that, IIS supports two more authorization mechanisms:

- Web permissions, which allow IIS to mark web files as read only, write, read, and write, and perform specific tasks like restricting script and .exe file execution. In the case of a web permission and a ACL conflict, the most restrictive applies.
- IP address and domain name restrictions: you can grant access to all hosts other than those you specifically deny or make sure that no host has access other than those you specifically allow

e. Custom Authentication and Authorization

ASP can implement custom authentication and authorization using a mechanism similar to the previous described JSP custom login system mechanism. The difference is that instead of a login page that receives the user credentials and a servlet that makes the necessary checks, in the ASP case, there is only one ASP page that receives the user credentials and makes all the checks. All other pages of the application are protected from direct access, checking the session object for valid login id (a server side include (SSI) file does exact this task). See my actual ASP implementation for details. However, Microsoft highly recommends the use of the built-in authentication and authorization methods instead of the custom ones.

B. CONCLUSIONS

I think that we cannot directly compare ASP and JSP technologies in the context of web-security. Most of these issues are handled by the underlying web-server or the operating system. ASP relies on IIS, which is a production-strength web server, along with Windows. JSP defines a standard API that must be implemented from a JSP container. JSP containers, like Tomcat, can be mounted in other more robust web servers like Apache or IIS. There are also some new web servers that have built-in JSP and servlet support, providing production strength and robustness.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS

This thesis explored web-based database development with ASP and JSP. Both are among the leaders in web technology. They both created as an alternative to the CGI programming. Among their goals were to:

- speed-up the overall web-application development
- separate the data presentation from the application logic
- facilitate the web page - database interaction

Up to a point, both are successful in their goals. However, there are also important differences between these two technologies. The objective of that thesis was to find these differences and evaluate them.

One of the most important differences is the platform independence. While ASP technology has to be used with Microsoft Windows and their web servers (there are few third-party utilities that allow ASP to run in other platforms like UNIX, but they are not widely used), the JSP technology is platform independent, and can be used in Unix, Linux, and Windows as well. Some of the current industrial-strength servers provide built-in support for JSP, like iPlanet and WebSphere, while others like Apache or Windows IIS can be configured to use JSP easily. This is definitely an important factor if we want our web application to be able to be up and running in different machines with different operating systems.

Issues related to application architecture are also an important difference. JSP allows much more flexibility, scalability, and code reusability through a number of web components like servlets, JavaBeans, and Enterprise JavaBeans. All these components can be incorporated in a JSP web application in order to help developers and designers to separate presentation, control, and application logic tasks. This greatly facilitates maintenance, extends application lifecycle, and permits already developed and tested components to be reused. ASP technology doesn't provide that variety of web components, relying primarily on COM objects as its web components. However, developing COM components is not an easy task, and needs more effort than developing JavaBeans or servlets.

The underlying language is definitely an important factor. ASP supports scripting languages like VBScript. These are usually a subset of other compiled languages. As a result they are somehow limited comparing to a compiled, full-featured language like Java, which is supported by JSP. Moreover, scripts created with scripting languages like VBScript, are parsed dynamically each time the script is requested (ASP case), resulting in extra overhead. On the other side, scripts created with a compiled language like Java (JSP case) are parsed once, compiled and stay in memory, ready to serve immediately any future request. It is not accidental that in .NET, the new platform of Microsoft, a new compiled language was introduced, the C#, which has a lot of common features with Java (creates bytecode when it is compiled, built-in memory management, etc), and is going to be used with the new version of ASP the ASP+, based on the .NET platform.

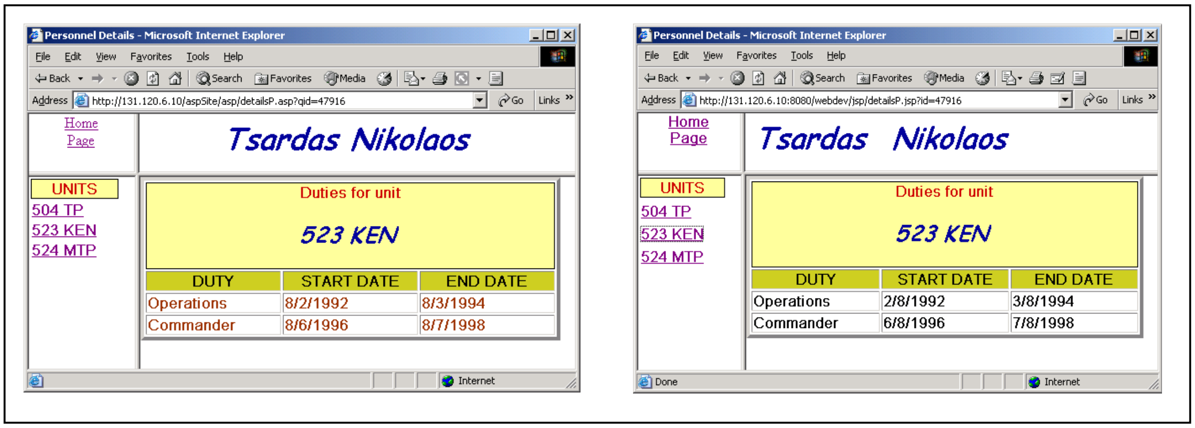
Nevertheless, we have to mention that with ASP is easier and faster to build small or medium web applications due to its immediate integration with the IIS, the Windows web server and the built-in support for a number of available web objects like Microsoft's Active Data Objects (ADO). Because ASP technology is in the market much more time than JSP, there is more gained experience among ASP developers than JSP ones, and also there are more tools currently available for web application development with ASP than with JSP.

Finally, the decision between the two technologies will be greatly affected by the current expertise of our development team and the specific requirements of the application, which will be developed.

Appendix A

A. APPLICATION DESCRIPTION

In this appendix, two web applications (prototypes) will be presented, one with ASP and another with JSP technology. Both applications are used to manage and interact with the same database. The database stores information about military personnel, units, and duties. Also, both applications have the same layout, user interface and functionality, as we can see in the sample pages in Figure 13.



Personnel Details - Microsoft Internet Explorer

Address: http://131.120.6.10/aspSite/asp/detailsP.asp?qid=47916

Tsardas Nikolaos

Duties for unit

523 KEN

DUTY	START DATE	END DATE
Operations	8/2/1992	8/3/1994
Commander	8/6/1996	8/7/1998

Personnel Details - Microsoft Internet Explorer

Address: http://131.120.6.10:8080/webdev/jsp/detailsP.jsp?id=47916

Tsardas Nikolaos

Duties for unit

523 KEN

DUTY	START DATE	END DATE
Operations	2/8/1992	3/8/1994
Commander	6/8/1996	7/8/1998

File Edit View Favorites Tools Help
Back Search Favorites Media
Go Links
Home Page
UNITS
504 TP
523 KEN
524 MTP
Internet
Done

Figure 13. Identical ASP and JSP Pages for Database Information Retrieval.

However, the underlying application architecture is totally different. The ASP application consists of just eight ASP files. Each such file has to receive user requests, connect and query the database, and finally present to the user the requested information. Active Data Objects (ADO) are used for any interaction with the database. There is no other type of web component, beyond these ASP pages. Specifically, the application logic and the presentation tasks are implemented as follows:

- One ASP page is responsible for the user authentication. It is called only when the user accesses for first time any page of the application. Then, it queries a table in the database, which keeps all the user credentials. If the submitted user credentials match any stored credentials, then access is granted. Otherwise, access is denied, and the user is prompted to try again.

- One ASP page is used as a homepage of the application. It presents to the users the title of the application and the main menu.
- One ASP page is used as a search page. It allows users to search the database for a person based on the person's ID or last name. Then, the page allows the user to double click any retrieved person in order to view more information about the person's unit and duties.
- Four ASP pages are used to present to the user information about the units and duties of a particular person. Three of the pages are used inside three frames presenting the actual data. The fourth page includes these three frames, without any presentation task, and it is the actual ASP page that is called from the above search page.
- One ASP page is used for the database maintenance. It allows the user to search, navigate, insert, modify, and delete records from the database.

On the other side, JSP application uses a number of web components beyond JSP pages like servlets, JavaBeans, and regular Java classes. Each type of component has specific tasks. JSPs are used only for presentation purposes. A servlet is used only for the information control flow. JavaBeans and a number of Java classes are used to implement the application logic and to interact with the database. The entire application consists of nine JSP pages, one servlet, three JavaBeans, and seventeen regular Java classes. We can see that the number of files that the JSP application consists of, is much larger that the ASP equivalent, due to the different application architecture and the discrete separation of tasks. Specifically, the application is implemented as follows:

- All the JSP pages are used to present to the user a number of web pages such as the application's homepage, search page, view pages, and a page for the database maintenance. These pages are identical to the ASP pages we saw above (from a presentation point of view). They accept the user requests and forward them to the controlling servlet. Also, they receive back processed data from the same controlling servlet and present it to the user.

- The controlling servlet accepts all the user requests (search, view, insert, modify, delete) through the JSP pages. Then, forwards each user request to the appropriate Java class for processing (using a hash table), according to the type of request. When the processing is done, the response is returned to the servlet, in order to forward it to the appropriate JSP page for presentation.
- The JavaBeans are used to encapsulate the results of the database queries. They are created by regular Java classes that handle the interaction with the database. Then, they are forwarded, via the controlling servlet, to the JSP pages for presentation.
- The regular Java classes are used to implement the application logic. There is a singleton class that performs all the necessary database interaction. That class extends another class that handles the connection pooling. The details about the database issues and the connection polling are presented in the following subsection. Also, for any single action that the application performs there is one small class that does exactly that. For example, the application lets user search for a person based on his ID, so there is a Java class named FindPersonById.java that does exactly that. That class can be reused in other parts of the application where the same functionality is needed (to search a person by his ID). Additionally, if any modification is needed in the future, that modification can be isolated in a small class, without affecting other parts of the application.

B. DATABASE CONNECTIVITY ISSUES

Both applications retrieve, store, update, and delete data from a database. The MS Access 2000 was used as a database. With not much effort, we could use any other database, more robust than Access, like MS SQL Server or Oracle. The ODBC for ASP, and the JDBC for JSP allows that transition, simply by using an appropriate driver and minor modifications in the application’s code. Two types of database connections were used:

- Local, meaning that the application was in the same machine with the database
- Network, meaning that the application was in a different machine than the database, but in the same Local Area Network (LAN). A Fast Ethernet 10/100 LAN was used.

Additionally, connection pooling was used in both applications. One of the most time consuming processes is the connection with the database. In the case of the Internet, where the users will not wait much time to view the requested data, that kind of delay is a significant issue. Connection pooling comes as a solution to that problem. In that case, a number of connections are established initially, when the application starts. The number of connections depends on the number of anticipated concurrent users, the nature of the application, and the hardware and software limitations. Then, every subsequent database connection request will be served by the already established connections, resulting in better performance. However, each technology implements connection pooling differently. In the JSP case, regular Java classes were used to establish a number of initial database connections, making them available for future requests. The type of connections and the way that they are used by the application is totally configurable via the Java classes that implement the connection pooling. A Java class from *Core Servlets and JSPs* book, [Ref. 7], was used for the JSP connection pooling implementation.

On the other side, ASP implements connection pooling using a Windows built-in feature. Specifically, the Windows administration tool *Data Sources (ODBC)* allows the connection pooling to be set for every supported database driver (an account with administrator privileges required). However, this implementation is somehow limited, because it does not establish a number of connections up-front, but instead it extends the life of the connections that will be established when the application will start running, allowing these connections to be reused. This is transparent to the developer, and it is handled by the Windows. However, in Chapter V "ASP-JSP Performance" we saw that JSP connection pooling implementation, outperforms the Windows's one.

LIST OF REFERENCES

1. Buyens, J., "*Web Database Development*", Microsoft Press, 2000.

2. Eddy, S., "*Active Server Pages 3*", IDG, 2000.

3. Fleet, S., "*Active Web Database Programming*", SAMS, 1999.

4. Howard, M., "*Designing Secure Web-Based Applications for Windows 2000*", Microsoft Press, 2000.

5. MCSE, "*Internet Information Server*", Sybex, 2000.

6. Hanna, P., "*JSP, The Complete Reference*", MacGraw-Hill, 2001.

7. Hall, M., "*Servlets and JavaServer Pages*" Prentice Hall 2000.

8. Hunter, J., "Java Servlet Programming*",* O'Reilly, 2001.

9. Fields, D. "*Web Development with JSP*", Manning, 2000.

10. Williamson, H., "*HTML Master Reference*", IDG BOOKS, 1999.

11. Wilton, P., "*JavaScript*" WROX, 2000.

12. Reese, G., "JDBC and JAVA", O'Reilly, 2000.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Chairman Code CS
   Naval Postgraduate School
   Monterey, California

4. Professor Thomas Wu, Code CS
   Naval Postgraduate School
   Monterey, California

5. LCDR Chris Eagle, Code CS
   Naval Postgraduate School
   Monterey, California

6. D.I.K.A.T.S.A
   Inter-University Center for the Recognition of Foreign Academics Titles
   Athens, GREECE

7. Nikolaos Tsardas
   Ptolemaida, GREECE